Explorative Structural Design

by

Rory P. Clune

B.Eng. (Civil) – University College Cork, Ireland, 2008

Submitted to the Department of Civil and Environmental Engineering in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Civil and Environmental Engineering
at the Massachusetts Institute of Technology

June 2010

Signature of Author: _____
Department of Civil and Environmental Engineering
May 7, 2010

Certified by: _____
Jerome J. Connor
Professor of Civil and Environmental Engineering
Thesis Supervisor

Certified by: _____
John A. Ochsendorf
Associate Professor of Civil and Environmental Engineering and Architecture
Thesis Supervisor

Certified by: _____
Denis Kelliher
Lecturer in Civil and Environmental Engineering, University College Cork, Ireland
Visiting Research Scholar of Civil and Environmental Engineering
Thesis Reader

Accepted by: _____
Daniele Veneziano
Chairman, Departmental Committee for Graduate Students

Explorative Structural Design

by

Rory P. Clune

Submitted to the Department of Civil and Environmental Engineering
on May 7, 2010 in Partial Fulfillment of the
Requirements for the Degree of Master of Science in
Civil and Environmental Engineering

ABSTRACT

The thesis proposes a new way of thinking about structural design software. The current state of computational structural design in practice is assessed, and a review of relevant literature and existing software identifies both the strengths of existing approaches and areas in which contributions can be made. A new approach is proposed which combines the strengths of architectural modeling software with structural analysis software, and an original object-oriented framework for the development of next-generation structural design tools is presented.

The thesis shows that the field of structural optimization, long maligned by engineers for its impracticalities for engineering practice, can be made relevant and beneficial in providing techniques to explore the design space in an optimally-directed way, leading to the discovery of unexpected and novel structural designs which are easier to build, use less material, and cost less than structures designed by conventional software. The software framework is extended to include these optimization components and to facilitate the future inclusion of new algorithms by users. A fully functional design environment is developed and presented as an implementation of the work of the thesis.

Thesis Supervisor: Jerome J. Connor
Title: Professor of Civil and Environmental Engineering

Thesis Supervisor: John A. Ochsendorf
Title: Associate Professor of Civil and Environmental Engineering and Architecture

Acknowledgements

I wish to thank Professors Jerome Connor and John Ochsendorf for their advice and invaluable insight in supervising this work. Their sustained enthusiasm for, and interest in, the work was a source of great encouragement. Thanks to Dr. Denis Kelliher for his advice and help, and for his most notable generosity with his time and computational expertise.

Without the enduring long-term support and encouragement of Deirdre and Conor Clune, this work, and my time at MIT, would not have been possible. I thank them both sincerely.

I am grateful for the support of the MIT Edward H. Linde Presidential Fellowship, the MIT Logcher Traveling Fellowship, and of the MIT Department of Architecture's Marvin E. Goody Award.

Rory Clune

Table of Contents

List of Figures

Chapter 1 Problem Statement

1.1 Good Design

Much has been written on the subject of *good design*. In any field, good design is an inherently difficult concept to define, not least because it is highly subjective. From project to project, and from designer to designer, the criteria for evaluating design and the relative importance of these criteria vary significantly. The German industrial engineer Dieter Rams (1932 -), originally trained as an architect, expressed his "Ten Principles for Good Design" in the early 1980s (Kemp and Ueki-Polet, 2009). These clear and universally applicable set of criteria are:

- *Good design is innovative*
- *Good design makes a product useful*
- *Good design is aesthetic*
- *Good design makes a product understandable*
- *Good design is unobtrusive*
- *Good design is honest*
- *Good design is long-lasting*
- *Good design is thorough down to the last detail*
- *Good design is environmentally friendly*
- *Good design is as little design as possible*

Although articulated by an industrial product designer, all of these statements are applicable to the design of large-scale civil structures. They should not be taken as a strict definition of what qualifies a structural design as good. Rather, they are an instructional reminder that designers should resist evaluating a design based on a single measure, or on the achievement of one particular goal. The vast majority of design problems are multi-objective (Tapabrata, 2004), and these multiple objectives are unlikely to be firmly set in stone, even in the late stages of the design process. This observation is a major motivation for this work, and will be revisited.

1.2 Problem Statement Identification

In the building and construction industries, there is often a clash of ideals between architects and engineers. The two practices have different preconceived notions of what is most desirable and important in designing a structure, and designers trained in either practice will usually adopt very different approaches (Lawson, 1980). This has been observed by the author in interaction with practicing structural engineers and architects, and in collaboration on academic design projects with architecture students.

Briefly, architects concern themselves with the aesthetics and context of a structure, as well as various interpretations of the utility it provides to its users. Engineers are more concerned with the physical rationality of the design; with the goal of designing a structure that efficiently fulfills a predefined purpose at a low cost.

Despite these differences, there exist numerous clear examples of good and bad structural design. Figure 1.1 (a) is an image of Grimshaw and Partners' Waterloo Terminal in London, the structural design of which was carried out by Anthony Hunt Associates. The design of the roof provides an efficient and elegant solution for a long span. On the right is a section of Terminal 2E in Paris' Charles De Gaulle airport, whose structural design team was lead by its chief architect Paul Andreu (Bolton, 2005), and which collapsed shortly after being built in 2004.

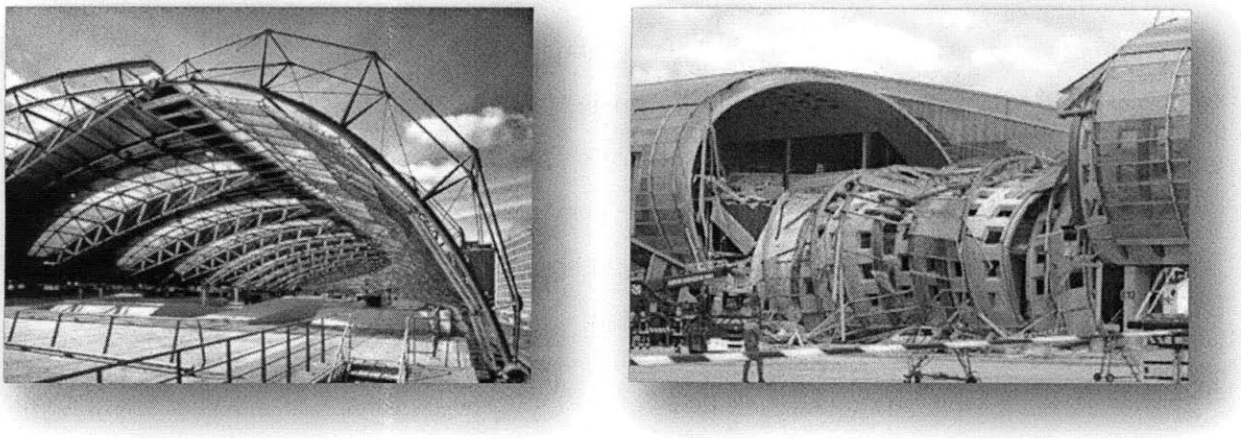


Figure 1.1 a) Good Structural Design **b) Bad Structural Design**

In practice, computer programs which were primarily intended to perform structural analyses are used as design software. A sequential, iterative approach is used by most designers. An initial form is defined (usually by the architect), and an engineer models this design using some finite element method (FEM)-based structural analysis software. He assesses the output from this software to evaluate the design and, based on this evaluation, modifies some parameters. The analysis is run again, and the iterative process continues until an acceptable solution is reached. There will inevitably be some give and take between the architect and the engineer, as the modifications to the design made by the engineer, presumably in the interests of structural performance, may not be acceptable to the architect or may cause him to reconsider his initial design. This is not to say that architects and engineers have fundamentally different and irreconcilable intentions during design. The differences in specialty and training between the two fields, however, will almost inevitably necessitate compromise of some sort. The development of design tools which encourage engineers and architects to explore the common ground they share as designers is a credible way to improve the structural design process.

Even if the form of the structure has been defined, and the position of primary members identified, the engineer is still faced with the challenge of member sizing, a complex design problem in itself. While FEM solvers can give detailed analysis of how a given configuration of member sizing behaves, it is not at all clear how to arrive at a first iteration of such a design. Typically, the engineer uses past experience and heuristic 'rules of thumb' to arrive at this first iteration. Final designs often do not deviate greatly from this first version, which is by no means necessarily close to optimal. Given the particular success of

sizing optimization algorithms, (Vanderplaats and Moses, 1972), it is surprising that they are not typically applied at this stage of the design process.

The initial design may be determined by hand, or by using a geometric modeling or computer-aided design (CAD) program. Most geometric modelers have no analysis capability, and so a method of transferring the representation of the structure from the modeling software to the analysis software must be found. Although a selection of geometric modelers and CAD programs now have built-in analysis capabilities, there are both merits and shortcomings to many of the implemented approaches.

On the positive side, analysis software has become highly sophisticated, providing reliable and accurate simulations of real structures. Geometric modelers allow the designer freedom of expression to quickly and conveniently explore a wide range of possible geometries. However, the analysis software with which the engineer spends most of his time has little or no design sensibility. While it makes the structural feasibility of designs much more likely, it does little to encourage good design – the designer becomes overly focused on the initially defined solution, and because of this cannot adequately address the aesthetics and context of his design. This is known as "design fixation" (Purcell and Gero, 1996). The potential for greater efficiency, both structural and economic, is greatly reduced by considering such a narrow portion of the potential design space.

Conversely, the starting point for the described iterative procedure is often arrived at without due consideration of structural behavior and rationale. The explorative philosophy of geometric modelers leads the designer to be so unrestrained as to produce designs which, though aesthetically and contextually pleasing, create unreasonable difficulties for the engineers and builders entrusted with their realization. As an example, the software packages of *Rhinoceros®* (McNeel, 2007) and *CATIA®* (Dassault Systemes, 2002) have enabled much more complex geometries to be created, but the structural, environmental, and economic logic of these geometries is often missing in design development.

The academic structural optimization community has developed a plethora of algorithms and tools to automate the design of structures (Haftka and Gurdal, 1992). Virtually every type of known optimization solution method has been applied to some type of structure, with varying degrees of success. A number of commercial structural optimization packages are available on the market, and some popular analysis packages now have built-in optimization capabilities, such as Vanderplaats R&D's *Genesis* (Vanderplaats R&D, 2009) and Altair's *Optistruct* (Altair Engineering, 2010).

In searching for what it determines to be the "best" design, a structural optimization algorithm explores a vast range of possibilities faster than any human designer possibly could. Surprising and novel designs often emerge. These may not be intuitively "good" in some sense to a designer, and as such would probably not have emerged without optimization, even from an exhaustive manual design process.

The results of structural optimizations are often dismissed. The most common cause for this is a lack of practicality or of constructability, both of which are difficult, if not impossible, to express mathematically. Structural optimization algorithms generally concern themselves with the pursuit of a single global optimal solution defined by a single objective, such as minimum weight, and therefore do not exploit their potential to help a designer to explore all areas of the design space in an optimally-directed way (Cohn, 1994). Significantly, the definition of this global optimum, if it is to include such subjective objective categories as aesthetics and constructability, will necessarily vary from designer to designer.

The goal of this work is to produce a structural design tool that will combine user intuition and experience with mathematical optimization to encourage creative exploration in the seeking of design solutions.

In the pursuit of good structural design, it is desirable to reach a compromise of sorts. On one hand, creative exploration of possibilities and alternatives throughout the design process is shown to be crucial to arriving at good solutions on a consistent basis (Von Buelow, 2008; Gero, 1994). On the other, a designer should seek to constrain such a thorough exploration to the realm of engineering feasibility, and should be able to guide the design process to as small or large an extent as desired.

This work addresses the development of a structural design computer program which engages the designer's experience and intuition by allowing him to creatively explore potential solutions while receiving real-time feedback on structural performance from the included analysis methods. While this alone will encourage the designer to creatively explore designs to a greater extent than traditional finite element packages, the algorithmic components of the program, in the form of structural optimization algorithms, will provide a second rich source of creativity. Attention is given to the value of capturing past design solutions for similar problems, and to methods which enable these solutions to be freely loaded from memory, and seamlessly analyzed, modified and optimized in a single program. These four elements are the key contributions to this explorative design tool.

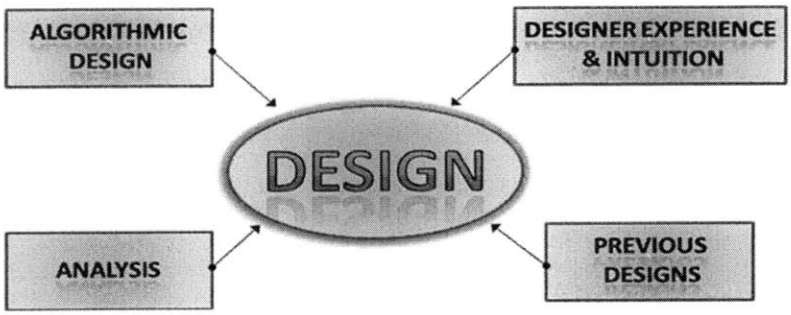


Figure 1.2 Components of the explorative design process

To develop a program which fundamentally rethinks the way in which engineering software should be designed, the work begins from a clean slate. No existing software is used to build the architecture of the system, the foundation on which it is developed. A thorough embracement of the philosophy of Object Oriented Programming (see chapter 2) is used, free from the restraint of conforming to existing practices in developing modeling or analysis software.

1.3 Thesis Structure

The rest of the thesis is divided into four chapters. Chapter 2 is a literature review, divided into two distinct sections. The first examines the application of Object Oriented Programming (OOP) to structural mechanics software and the second examines the state of structural optimization. Both sections focus in particular on contributions to structural design, and identify shortcomings in the literature where future work should be done. Chapter 3 describes the motivation behind, and the technical specification of, an object oriented architecture for structural design software which is designed from scratch. Chapter 4 extends this architecture to include optimization algorithms, identifying how this particular approach to optimization overcomes some of the limitations faced by optimization at the conceptual design stage.

Finally, chapter 5 presents the results of this work. A structural design program called *Immersed Design,* built using the theory developed in chapters 3 and 4, is presented. Structural designs developed using the software are displayed. The original contributions of the thesis, including improvements over existing computational design methods, are clearly stated in this final chapter.

Chapter 2 Literature Review

2.1 Introduction

This chapter charts the progression of two distinct fields of research, with a focus on their contribution to structural design. Within the field of structural mechanics, the chapter examines the application of the Object Oriented Programming (OOP) paradigm to the design of structural analysis software. As it has been the subject of the most research, the application of OOP to Finite Element Method (FEM) computer programs is first examined. This is followed by a review of work which has been explicitly directed at the use of OOP to improve the design capabilities of engineering software. The second distinct review is of the field of structural optimization, with a focus on creative design and synthesis of civil structures.

2.2 Object Oriented Programming in Structural Mechanics

Object Oriented Programming (OOP) is a computer programming approach which aggregates data, and procedures which operate on those data, into entities known as objects. The packaging of data and functions in this sense is known as encapsulation. Its origins can be traced to the development of the Smalltalk™ language by the Xerox Corporation in the 1970s. Its emergence as the dominant modern computer programming methodology is commonly thought to have occurred with the rise in popularity of the C++ language in the mid-1990s. Nowadays, the object-oriented Java and C# languages are widely used (Booch, 2007).

In OOP, objects can interact with each other, and can be manipulated by invoking their methods. The OOP paradigm can avoid and mitigate many difficulties which arose in the software development process prior to its emergence (Pressman, 2001), and can improve the reusability, maintainability and modifiability of code.

The blueprint from which an object is created is known as a class. An object is therefore an instance (or instantiation) of a class, which specifies the characteristics of the object. As in the real world, many distinct objects, even if identical, can be instantiated from a single blueprint. Along with encapsulation, two key features of OOP are inheritance and polymorphism. Once a class has been defined, a second class can be defined which inherits all the characteristics of the first. In this case, the first class is referred to as a parent (or base) class, and the second as a child (or derived) class. Simply put, the child class can do everything that the parent class can, without the need for any further programming. Polymorphism is a concept which allows a programmer to refer to an object and to run its procedures without knowing precisely what type of object it is. As long as the object is an instance of a class which derives from a known parent class, the programmer can treat the object as if it were an instance of the parent class. A single line of code could therefore apply to numerous different object types, a feature which is extremely useful in large, complex programs (Schlaich, 2006)

2.2.1 Object Oriented Programming of the Finite Element Method

Among the various structural mechanics methods, the Finite Element Method (FEM) has seen the greatest amount of research into, and application of, OOP techniques. Given FEM's near-ubiquity as an analysis tool for the structural engineering community, this is unsurprising. The advantages of using OOP for structural mechanics, with a particular emphasis on FEM, have been well identified (Cross et al., 1999). Although these advantages are numerous, the fact that OOP is a natural way to manage and to logically organize highly complex software is often identified as the strongest (Mackie, 2001a).

The purpose of this section is to examine the programming concepts and techniques which have been applied to FEM programs, as well as the advantages they offer. The review of these apparently abstract concepts in the literature is necessary to identify the best practices in structural software development, so that they can be considered in the development of a new structural design tool.

OOP has been used in the design and development of structural analysis software since the early 1990s, when fairly straightforward implementations of finite element programs emerged (Forde et al., 1990; Mackie, 1992). These implementations departed from classical programming techniques by creating separate objects for components of FEM models such as degrees of freedom, nodes, supports and elements. At a time when the benefits of modern object oriented languages had not been fully realized, such early works were effectively a rewriting of previous FEM codes in the object oriented format. They were the precursors to many advances that followed, but did not significantly change the developed software.

Throughout the 1990s, further research was carried out, leading to greater leveraging of the benefits of an object oriented approach. The work of Menetrey and Zimmerman (1993) and of Dubois-Pelerin and Regon (1998) on nonlinear finite element analysis techniques provides highly illustrative examples of the reusability of code in an object-oriented environment. Both papers describe the development of nonlinear finite elements as an extension of pre-existing linear elements, without the need to rewrite code which is common to the two. Although these papers are specific to FEM, they are a practical example of code reusability in the development of a structural mechanics program.

Of greater interest in the context of this thesis is the use of objects to capture features of structural mechanics which are not unique to FEM. One of the first challenges to be addressed in this fashion was the integration of structural modeling and structural analysis components in a single software package, without the need to laboriously export data from one program and then import it into another. Abdalla and John Yoon (1992) describes a system that uses an object-oriented approach to integrate FEM and graphics software, representing both programs as distinct objects which sit within an overall master program. This strategy could be argued to be an automation of the data transfer approach, but it represented one of the first moves towards using an object-based approach to developing the underlying structure of engineering software.

Heng and Mackie (2009) give an overview of the software design patterns that have emerged in the design of FEM programs using OOP. A design pattern is the abstraction of a recurring solution to a design problem, formalized by Gamma et al. (1995). These patterns are a useful way to improve software quality and reduce development time by allowing developers to quickly survey the most widely and successfully applied strategies in a particular field. Within the realm of OOP, a design pattern aims to capture a map of the objects which populate the system, their key characteristics, and the interactions that occur between the objects.

Of the design patterns presented by Heng and Mackie, two are particularly relevant to the development of engineering software in general. The first of these is *Model-Analysis Separation*. The essence of this pattern is that objects related to the modeling of the structure, such as elements, nodes, and supports, should be separated from analysis objects, whose purpose is to formulate and solve a system of mathematical equations describing the structure. The methods and procedures to analyze forces on a structure should not be contained in the same objects that model the structure. This pattern recognizes that analysis objects operate on model objects, and implies that the group of analysis objects should be dependent on the group of modeling objects. Heng and Mackie's interpretation of this pattern is that model objects are "stable relative to analysis objects". The analysis system of objects should be amenable to changes, while the modeling classes are more likely to remain unchanged.

The model-analysis separation pattern can be seen in Patzak and Bittnar (2001), which structures an FEM code with a clear separation between objects containing the overall analysis methods for solving the model, and the modeling classes themselves. An intermediate "Engineering Model Class" is used to map information from the model subsystem to the analysis subsystem, and to manage the distribution of analysis results from the analysis subsystem back to the model objects. Rucki and Miller (1998) also recognize the need to separate modeling and analysis classes, and make a strong case for implementing the design pattern to enable future extension of the software. The paper shows how the addition of a new analysis method, which could provide new information about structural performance, does not require the rewriting of the methods that model the structure; it requires only the rewriting of the methods that analyze the model of the structure.

The second design pattern of interest is the *Model-UI Separation* pattern. The motivation for the pattern, as explained by Heng and Mackie (2009), is the avoidance of a situation where user interface (UI) procedures such as the rendering of a component of the structure, or the creation of a new component via a graphical interface, are embedded in the modeling objects. This undesirable situation creates a series of "bloat[ed] model classes with incoherent responsibilities". The pattern advocates the creation of separate UI objects, which can render or modify the existing model based on user input. When implemented in conjunction with the model-analysis separation pattern, there should be absolutely no coupling of the analysis and UI subsystems. The authors recommend including a direct reference to an overall model object within the UI object, rather than placing an intermediate object between the two subsystems to handle information passing.

The *Observer* pattern specified in Gamma et al. (1995) can be used to make the model independent of the UI. In this scenario, the model broadcasts a message every time it is modified. The UI object picks up this message and renders a visualization of the model. The authors identify the main advantages of using the Observer pattern and the Model-UI separation pattern as a clear division of responsibilities between objects, and the fact that changes to the UI, such as the implementation of a new UI system or UI component; do not require any modification or rewriting of the model classes. This pattern is recognized as a variant of the *Model-View-Controller* pattern, which is well known among OOP developers. A description of the pattern and its application in the design of FEM software was proposed by Mackie (2001b)

A large body of work has been published on the application of OOP to FEM computer programs. While much of the work relates specifically to FEM modeling and analysis, many of the general concepts are applicable to structural engineering software in general, and indeed to almost any engineering software that involves modeling and analysis of a system.

Although there is much to be learned about good software design in a structural engineering context from this particular body of research, there is a need to explore how the developed theory can be applied to a program with a strong design sensibility. No researcher has thoroughly extended the object oriented concepts and software design patterns beyond application to FEM alone, to examine how they might contribute to software which fundamentally rethinks the way computational structural engineering should be applied to the conceptual design stage. There is an immediate need to apply the concepts deemed most beneficial to such a design-oriented program, and to define the role current FEM solvers might have in such an approach.

Much of the literature has focused on redeveloping FEM solvers using the object oriented approach. A primary goal of this thesis is to define an object-oriented architecture for structural software using these established concepts, but without the preconception that FEM is the best, or the only, analysis tool to use for structural design and analysis.

2.2.2 Structural Engineering Software in General

Early in the development of the application of OOP to engineering software design, it was realized that OOP could fundamentally transform the way structural engineering software is designed. Although primarily focused on FEM software, Miller (1991) provided the earliest clear statement of this realization, with clear implications for any type of structural software. This paper states that, before the introduction of OOP, finite element programs ran in batch mode, as they had been originally been developed in the 1950s and 1960s. In batch mode, the computer executes a series of sequential operations without any manual intervention.

This works for well-defined structural analysis problems, where the design of the structure has already been specified. Miller recognizes that the design process is, by definition, not well defined. It is a fluid and imprecise process which involves constant modification of a model, and which requires the production of analyses and results with each model modification. The paper does not go as far as achieving the goal of changing engineering software to equally target all stages of design of a structure rather than merely building a design program on top of an analysis method. However, it does recognize the benefit of doing so, and outlines the beginnings of how this goal might be realized, long before the development of an OOP language and readily available computer hardware capable of doing so.

The "comprehensive redevelopment of structural engineering software" envisaged by Miller in 1991, and the move away from the fundamental perspective of traditional structural analysis software, has yet to occur in any sense that has had a widespread effect on practicing engineers and designers, although progress has been made. The paradigm of achieving a form of interaction between various programs by having them work from shared data still exists. A batch process still prevails, both in the previously described sense of the internal computational mechanics, and in the user's interaction with the software. The user defines a model, analyzes it, and interprets the results. Whether this sequential process occurs within a program which integrates modeling and analysis programs, or whether it requires the user to export data from one program to another, it is still essentially a sequential process requiring the specification of a well-defined model, and a user-issued command to run an analysis.

Despite the addition of features to make things appear otherwise, the design process using these commercially available tools, which are the prevalent norm in structural engineering practices today, is fundamentally the same as it was when computational mechanics first impacted structural engineering

fifty years ago. Two examples of such software packages widely used in practice are *SAP2000®* (Computers & Structures Inc., 2010) and *ABAQUS®* (Dassault Systemes, 2004). There are numerous others in use, but all share more or less the same philosophy of batch processing, where the user defines a structure, presses a button to run an analysis, and is presented with analysis results. Recently, there have been instances of Computer-Aided Design (CAD) and other such modeling programs with integrated analysis capabilities, such as the integration of the analysis engine of *ABAQUS®* into the *CATIA®* (Dassault Systemes, 2002) modeling software. Such examples provide some of the benefits of Miller's envisaged "redevelopment of ... software" without any undertaking of such a fundamental redevelopment. Real extensibility of the software is generally not possible, tying the user to the functionality provided by the original developers.

Many of the benefits that could emerge from a fundamental object oriented development of design software from a blank slate cannot emerge from this approach taken by the engineering software companies. Among these benefits is the ability to easily extend software to include new modeling, analysis and visualization capabilities without any complicated reprogramming of the links between programs, and without a thorough knowledge of the data structure being used by the software.

An idea which has been articulated in the literature is the simultaneous use of multiple modeling techniques to model a physical artifact or process (Delalondre et al., 2010). This paper focuses on the relationships between multiple models of a single system, and the operations that occur in order for them to remain consistent. Existing simulation techniques are used, with each new components added by a developer conforming to a specified programming interface in order to interact with the rest of the system. Of interest to the design of large-scale structures in the "multiple fidelity model simulation", where a number of different models, each requiring varying computational effort and producing results of varying precision, model a single physical system. The work outlines the benefit of using multiple models of a physical system, but does not consider the potential benefits for the design of civil scale structures. In order to make this work more relevant to structural engineers working at the civil scale, more consideration should be given to directing this approach towards synthesis rather than analysis, focusing on the purpose that various types of models serve in structural design.

2.2.3 Real-Time Structural Software

As part of an effort to create software with a stronger design sensibility, there has been significant interest in developing 'live' or 'real-time' structural analysis applications. In these programs, user interaction, visualization of the model and structural responses, and computation all appear[1] to occur simultaneously. Response to interaction happens instantly, without the need for the user to explicitly run an analysis. These highly interactive programs are useful for developing user intuition of structural behavior, and are particularly helpful in exploring initial design possibilities. Papers emerged in the 1990s on developing interactive FEM programs (Mackie, 1998; Rucki and Miller, 1996). These both featured an OOP approach, and described strategies which would allow a large amount of flexibility and user control in the algorithms used to analyze a finite element model.

More recently, software packages with real-time structural analysis capabilities have become available, such as the *Dr. Software* range of products (Dr. Software, 1998), *Arcade* (Martini, 2001; Arcade, 2002) and *Active Statics* (Greenwold, 2003). These software packages represent a definite move away from the traditional model of engineering software, and are an effective means of providing continuous information to a designer as he makes design decisions. These tools free the designer from much of the tedium of operating the software, and allow him to better focus on the effects his design decisions have.

Dr. Software's main program is entitled *Dr. Frame.* It provides real-time display of structural responses, and is a powerful tool for exploring the effects of varying loading and a number of other parameters on a structure. It represents a vast improvement over traditional software in enabling users to understand the behavior of a given structure. Its primary weakness is that it is not as easy to modify the form of the structure, and to quickly understand the effects of this variation in form. This limits its effectiveness as a design tool.

[1] The use of the word 'appear' in this context reflects that a computer with a single processor can process only one operation at any given time. The high speed at which the operations required for interaction, computation and visualization occur makes it appear as if they occur simultaneously. In a computer with more than one processor, more than one operation could occur at a given time.

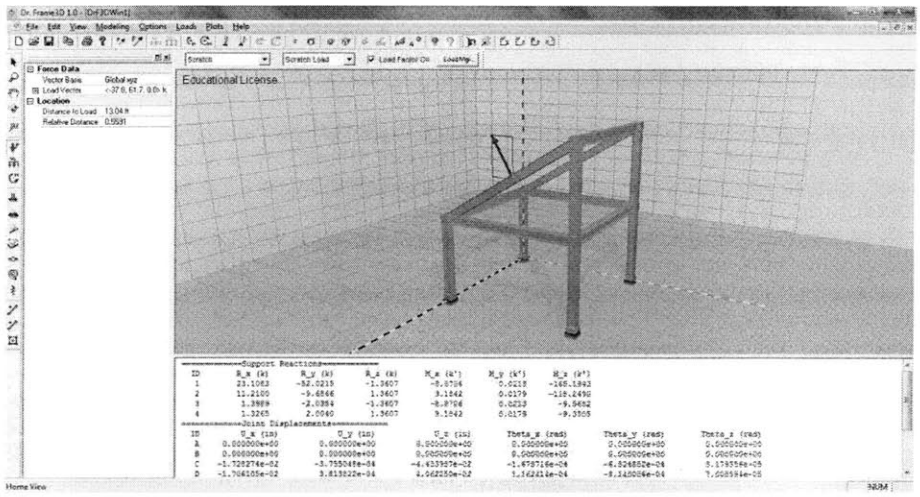


Figure 2.1 Screenshot of Dr. Frame (©Dr. Software LLC, 2002-2003)

Arcade uses a novel application of the particle-spring approach to model trusses and frames, a natural way to model nonlinear and dynamic behavior. These aspects of structural behavior, not typically captured by free or low-cost software, distinguish Arcade from many such software applications. Arcade is useful for developing structural intuition, but is in many ways subject to the limitations of a batch process. The user creates a structure and then runs a simulation. Some real time interaction is allowed, but modifications to the topology or geometry of the truss (other than deletion of members) requires a reformulation of the problem, and the user must exit the live simulation to redefine the structure.

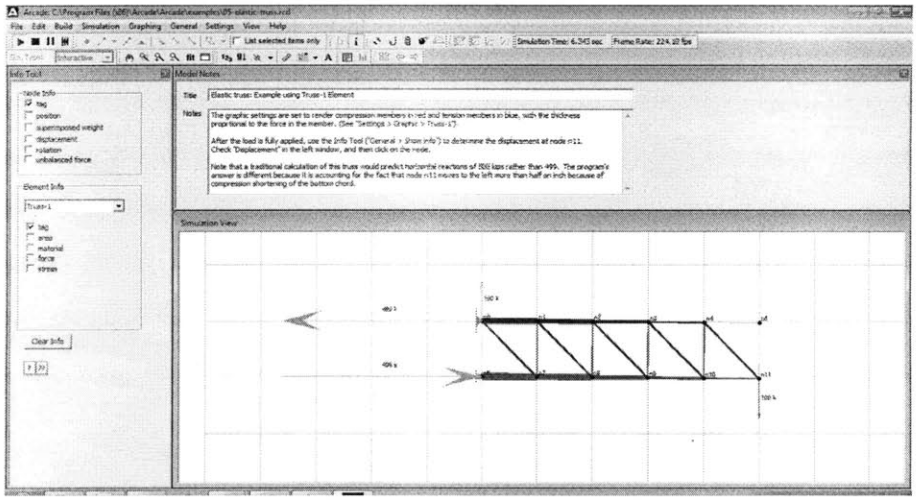


Figure 2.2 Screenshot of Arcade (©Kirk Martini, 2000-2009)

Active Statics uses the technique of graphic statics to analyze trusses and other structures which can be approximated by members carrying axial forces only. It has a strong design sensibility, in that the responses to changes in form and loading are graphically displayed in real time, allowing the user to explore a range of possible configurations quickly and intuitively. The software is limited to a selection of predefined topologies, and as such serves primarily as an illustration of the power of such an approach in a design context. The user cannot add or remove members from an existing model, and cannot create a model from scratch. As graphic statics is used, only statically determinate structures can be analyzed, limiting the applicability of the approach. Despite these weaknesses, Active Statics develops structural intuition via its interactivity, and has an overall design sensibility that developers of design software should seek to emulate.

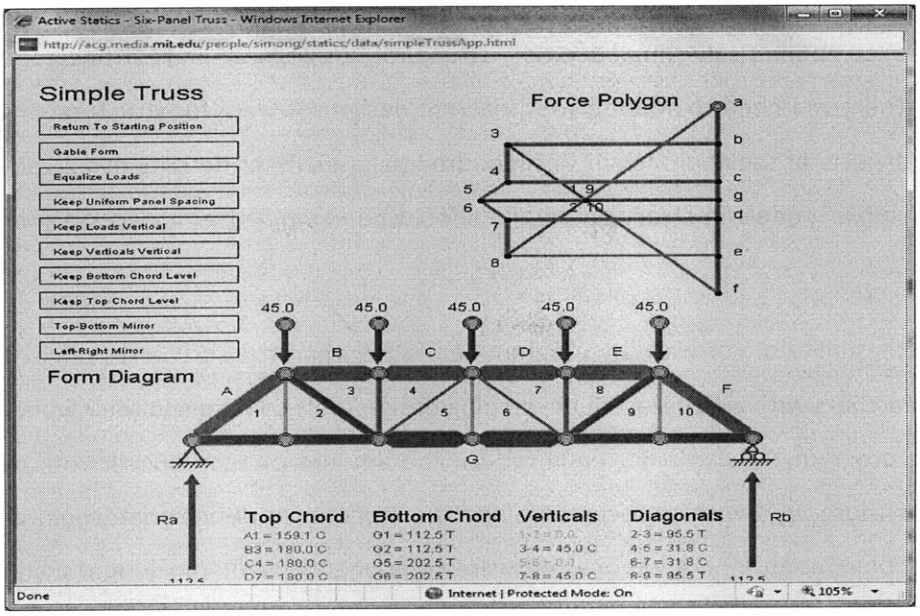


Figure 2.3 Screenshot of Active Statics (©Simon Greenwold, 2003)

Lindblad and Miller (2004) have further developed the idea of 'live' software by presenting a framework for a software environment that handles many design alternatives with real-time analysis capabilities. This work provides an automated management of many design alternatives, allowing for improved comparison between the alternatives for evaluation purposes. The authors describe methods to maintain relationships between designs, so that changes in one design will be reflected in all the designs

that derive from it in a hierarchical tree, removing the need to update every design possibility in the multi-design environment. The authors state that a design environment such as this should be able to perform differing degrees of analysis, using different solvers, on any subset of the designs or on all of the designs at the same time. The prototype software they describe also allows for an element of parametric design.

2.2.4 Necessary Future Work in the Object Oriented Approach to Structural Software

There is a need to develop a structural design program which has the strong design sensibility of a program such as Active Statics, but with the extended functionality required of a real-world engineering tool. Users should be able to create and modify and structure with ease, and a balance should be found between detailed numerical simulation of structural responses and strong design sensibility. Furthermore, there is a need to adopt an approach to design software for structural engineering which makes use of aspects of the approach of Delaldondre et al. (2010), as described in 2.2.2. The benefits of the use of multiple models in structural design should be examined as a necessary precursor to this application.

Should the extensibility of software by programmers other than the software's original developers be achieved, researchers with an interest in developing design tools could begin their work as an extension of an existing program. This would greatly reduce the amount of reproduced work by the academic community, as those who wish to focus on a specific aspect of a design tool would not have to first develop a bespoke design tool. Additional software components useful for design, such as optimization algorithms or improved user interfaces, could also be added without the need to develop an entire modeling and analysis program. To do so would require the definition of an appropriate software architecture intended specifically for structural engineering software, which incorporates the design patterns identified in works such as Heng and Mackie (Heng and Mackie, 2009) as well as aspects of Delaldondre et al.'s work (2010). This has yet to be done.

2.3 Structural Optimization

Structural optimization textbooks routinely credit A.G.M. Michell's landmark paper (Michell, 1904) with taking the first steps in the field (Hemp, 1973; Rozvany, 1989). Michell's work, conducted long before the age of digital computers, consists of an analytical treatment of framed structures. The paper identifies structures which solve a given structural design problem using a minimum amount of material, subject to a number of implicit assumptions (Rozvany, 1996). These have become commonly known as 'Michell Structures', and are still used today as benchmark solutions for many optimization methods (Rozvany, 1998).

The field of structural optimization has progressed significantly during the last four decades. The analytical treatment of framed structures was revived and extended, notably by Hemp and Rozvany (Hemp, 1973; Rozvany, 1989), among others. These works seek an analytical solution to an explicit optimization problem, usually with the goal of using a minimum amount of material to perform a given task, subject to a variety of constraints.

The application of optimization methods utilizing numerical techniques to civil structures dates back to the 1950s, when Heyman (1959) applied Linear Programming theory to the plastic design of frames. Numerical methods, more so than methods relying on an analytical approach, were widely studied and applied in the late 1980s and early 1990s (Bendsoe, 1995). However, the penetration of numerical optimization methods into industry has been more limited than this promising academic work would suggest (Haftka and Gurdal, 1992).

The field of structural optimization is subdivided in many ways. A commonly adopted division is to consider three classes of optimization problem: sizing (or cross-sectional), geometry (or shape), and topology optimization (Kirsch, 1989). Topology optimization seeks to optimize material layout within a design space, usually before any conceptual design for a structure has been defined. In structures which consist of discrete elements, topology optimization is concerned with the number and connectivity of these elements. Shape optimization addresses the geometric arrangement of a structure, assuming topological properties are fixed. Sizing optimization assumes that the topology and the geometry of the structure is defined, and seeks to optimize the cross-sectional properties of members, such as area, thickness or moment of inertia.

Variables in topology optimization problems are discrete, a fact which poses many intellectual challenges. The field can be divided into Layout Optimization (LO) problems, which deal with low volume-fraction grid-like structures, and Generalized Shape Optimization (GSO) problems, which deal with higher volume-fraction structures (Rozvany, 2001). Both problem types have been addressed analytically and numerically. The earliest approach to LO problems can be found in Michell's (1904) work on truss structures. Rozvany (1972) extended this approach to beam structures during the field's revival in the 1970s. A wide range of techniques have been applied to truss topology optimization problems, among them the Ground Structure approach (Hemp, 1973), Simulated Annealing (Shea et al., 2005) and Genetic Programming (Von Buelow, 2008). These are considered part of the LO subdivision of topology optimization. The latter two form part of a body of work known as stochastic optimization, which will be further explored in this section.

The optimization of the topology of structures having a higher volume fraction has been referred to as Generalized Shape Optimization since the early 1990s (Rozvany, 2001). The first method widely applied to this class of problem was the Homogenization Method, which enjoyed considerable success in the optimization of individual components (Bendsoe, 1995). The Solid Isotropic Microstructure with Penalization (SIMP) approach, developed in parallel by Bendsoe and by Rozvany and Zhou (1991), and the Evolutionary Structural Optimization (ESO) approach of Xie and Steven (1997) are two of the most widely applied GSO methods in use nowadays.

An example of the shape optimization of trusses can be found in Imai and Schmit (1981). Shape optimization is often combined with sizing optimization in the formulation of a single problem statement. An element of topology optimization is often included in shape optimization programs using the "multi-level design" approach outlined in Spillers (1975). In this approach, a continuous geometry optimization algorithm modifies a structure, and a series of heuristic rules are applied to modify topology. Usually, members which have become negligibly small or thin are removed, and nodes or points which have been moved very close to each other are combined.

Though a restricted class of problem due to the many designs possibilities that are ignored, sizing optimization has also been extensively studied and successfully applied (Vanderplaats and Moses, 1972).

Stochastic optimization methods involve an element of randomness. Modifications to the design are made based on a set of decision criteria which is usually not directly linked to the mechanics of the problem. The types of algorithms which show up most frequently in structural optimization applications

are inspired by natural phenomena, and modify designs in a fashion analogous to some occurrence in nature. Genetic algorithms (GA), based on the genetic theory of evolution, have been used to generate truss designs, simultaneously addressing issues of topology and shape optimization. Deb and Gulati (2001) used GAs in the optimization of truss design, and Von Buelow (2008) has used them in the pursuit of design exploration strategies. ESO (Xie and Steven, 1997), previously mentioned, is also a form of stochastic optimization. Stochastic optimization can be used for any of the three classes of structural optimization (topology, shape and sizing), although its natural handling of discrete programming, where design variables assume only values from a certain discrete set, makes it an inherently suitable way to handle topology optimization problems.

2.3.1 Structural Optimization and Design Creativity

Some researchers have explicitly identified optimization as a source of creativity in the design process. Shea et al. (Shea et al., 2005) use the stochastic approach of simulated annealing to simultaneously optimize topology and geometry in a developed program named *eifForm*. The method described generates a number of "optimally directed" solutions for a given design problem, along with performance statistics on each design, providing the user with a means to compare different designs.

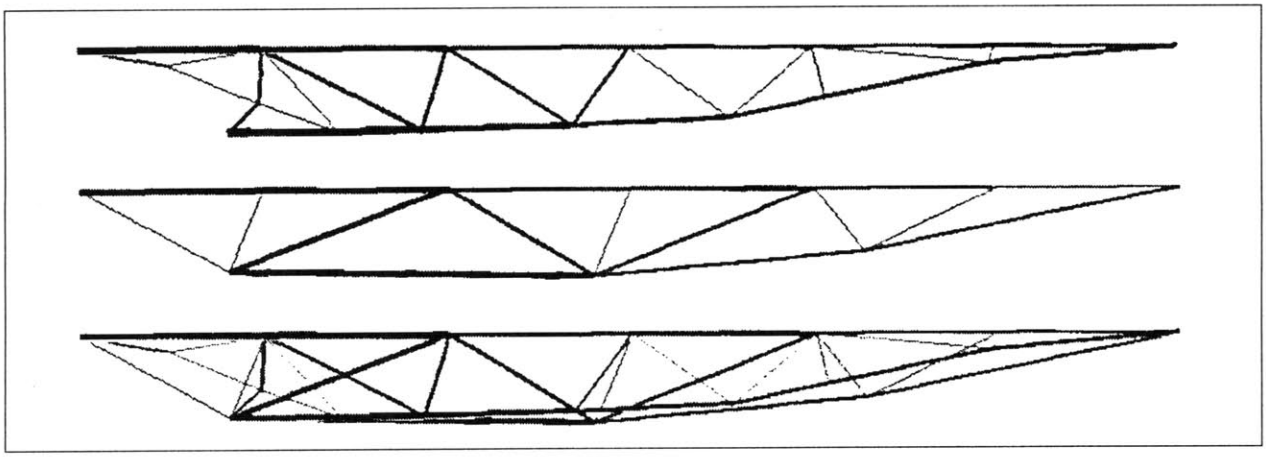


Figure 2.4 Cantilever truss alternatives designed using *eifForm* (Shea et al., 2005)

Von Buelow (2008) adopts a philosophically similar approach by using genetic algorithms to generate a population of "fairly good" solutions. A software implementation is described which allows users to visualize the various outputs in a manner conducive to comparison, and an "interactive mode" allows the user to guide the genetic algorithm's exploration of the design space to follow personal preference. Von Buelow shows how an exploration of multiple designs avoids "design fixation" early on in the conceptual design process.

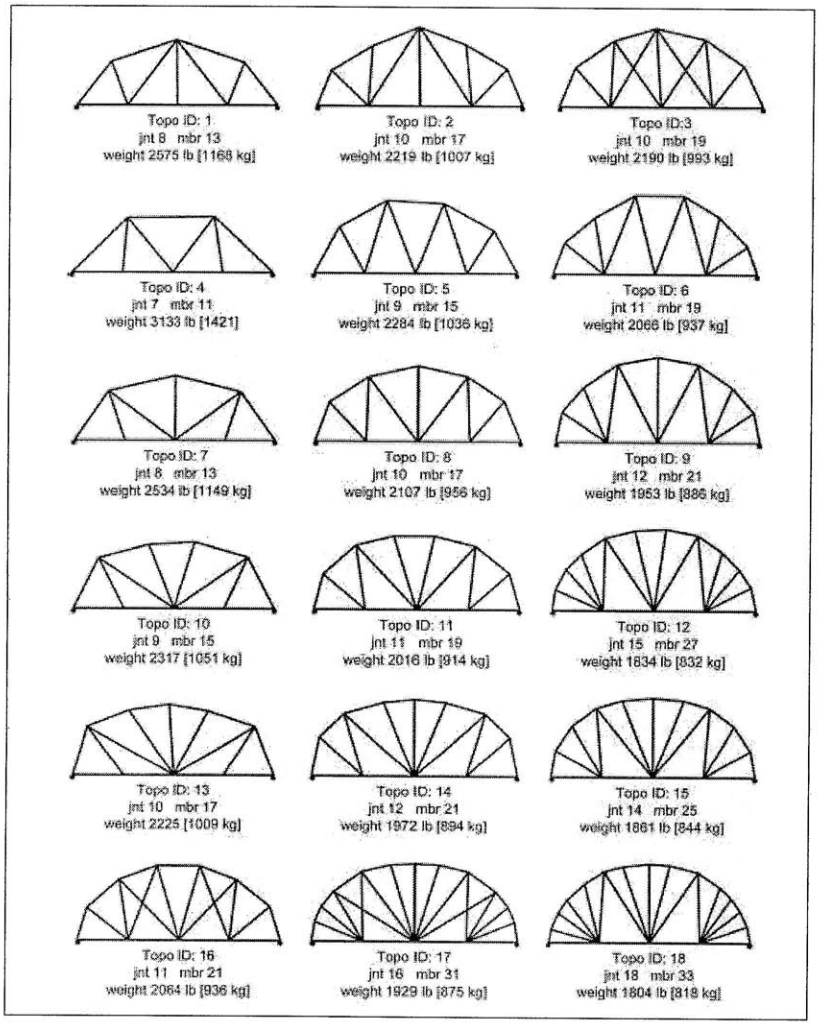


Figure 2.5 Selected results of truss design exploration using genetic algorithms (Von Buelow, 2008)

2.3.2 Structural Optimization and Real-World Considerations

In general, convex optimization problems are easier to solve than non convex ones (Yang, 2008). In a non convex problem, there may be many points which are locally optimum (they are better in some way than their close neighbors), but which may not be globally optimum (another better design may exist elsewhere in the design space). Much of the literature has focused on convex problems and on ways to reformulate or approximate structural design problems as convex. The convexity issue is a challenge in the application of optimization methods to real design problems, often formulated by designers with no optimization specialization. These real-world problems are often highly non convex.

A number of approaches to solve for global optima have been proposed (Floudas and Pardalos, 2003). An interesting approach involves the abandonment of the quest for a global optimum, and the pursuit instead of many "optimally directed" solutions from which to choose (Shea et al., 1997; Von Buelow, 2008). In a realm where a precisely defined optimization problem is unlikely to be representative of an inherently ill-defined and variable design problem, this seems like a sensible approach to take.

There are other factors which hamper the application of structural optimization methods to the design of civil structures. Much of the literature focuses on optimizing a structure to minimize or maximize a single objective, but real design problems are almost never single-objective (Tapabrata, 2004). A number of strategies have been presented to handle multiple objectives (Coello Coello, 1999; Miettinen, 1999), but none of these are generally adopted as a necessary prerequisite for any work in practical structural optimization. In addition, many of the potential objectives, such as aesthetics, context, or even personal preference for a certain design, are inherently difficult, if not ultimately impossible, to define in a computer program. This subjective class of design criteria distinguishes civil design problems from other structural design problems which are more amenable to an entirely mathematical treatment (in, for example, the aeronautics and automotive industries). Different designers attach different relative importances to different objectives, and some of these objectives are hard to express mathematically. Machine learning and other such fuzzy-logic based techniques have been used in an attempt to handle these difficulties (Thurston and Sun, 2008; Pullmann et al., 2003)

A similar issue is that many works of research seek to optimize a structure subject to a single load case, when most real structures are subjected to a wide variety of loadings. Again, much has been published on how to accommodate multiple load cases in an optimization process (Pritchard et al., 2005). In

general, the majority of published optimization methods are tested and evaluated with single load cases, which almost never occur in civil structures. An approach to the problem in Guest and Iguasa (2008) presents work on the optimization of structures accounting for variability in the applied loading, as well as accounting for the uncertainty of node positions due to manufacturing error.

These topics and considerations, which can be viewed as the extension of optimization programs from an academic context to real-world applications, usually appear at the end of a research paper, under the heading of "next steps". However, few researchers ever take these next steps. Many algorithms and approaches which have been shown to be successful under a narrow range of possibilities have yet to be shown to be applicable or inapplicable to the more general problems which inevitably arise in a design process.

2.3.3 Structural Optimization in Industry

Many major commercial finite element analysis programs include optimization capabilities, and companies that have traditionally focused on optimization methods have created products that include analysis methods. Vanderplaats R&D's *Genesis* program (Vanderplaats R&D, 2009) and Altair's *OptiStruct* (Altair Engineering, 2010), are two leading examples of programs which integrate optimization into the design process. Both programs have elements of topology optimization which allow for the automatic synthesis of new designs, and both have shape and sizing optimization capabilities which can optimize existing conceptual designs.

A number of less sophisticated, but freely available, structural optimization programs are available online. These are usually produced by universities and research institutions as a demonstration of ongoing work. One such example is the web-based *TOPOPT* program developed by the TopOpt group at the Technical University of Denmark (TopOpt Research Group, 2009).

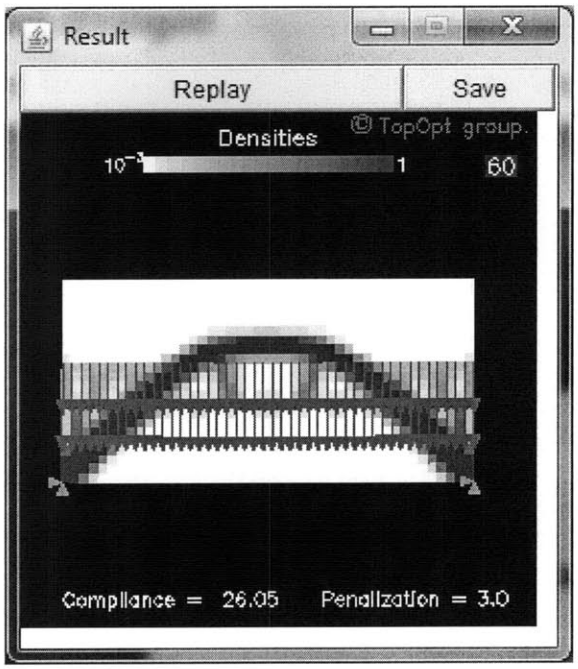


Figure 2.6 Screenshot of web-based TOPOPT (© TopOpt Research Group at the Technical University of Denmark)

2.3.4 Necessary Future Work in the Application of Optimization to Structural Design

In order to make the academic accomplishments and advances in structural optimization relevant to the building industry, there is a need to identify concrete reasons that have limited optimization's use by designers and to propose methods to resolve the limitations. Structural optimization problems are often approached without first considering their use by designers with little or no specialized training in optimization techniques, motivating an exploration of the integration of optimization into a design tool which designers are comfortable using.

A second gap in the literature exists in the lack of a defined, replicable method for optimization algorithms to link to modeling, analysis and user interface components of structural software. If a software architecture incorporating optimization algorithms and showing how they should interact with these other components was defined, it would provide a standard way for researchers to extend their own, or others', software by adding new optimization algorithms. A program with a well-developed user interface and modeling capabilities could, for instance, be extended by an optimization specialist and retain its previous favorable qualities. The aspects of a software architecture that would allow this sort

of extensibility in a structural optimization have not yet been described or implemented in the literature.

Chapter 3 Software Framework

3.1 Introduction

The goal of this chapter is to describe the framework for a structural design program which meets the needs identified at the end of chapter 2. Section 3.2 discusses models as abstractions of physical systems. Sections 3.3 and 3.4 motivate the use of multiple modeling techniques in the structural design process, and describe how the use of multiple models can give design sensibility to structural software. Section 3.5 provides a technical description of an object-oriented architecture for structural design software developed based on these principles. Techniques to enable extension of the software by third parties are also developed and described in this section. The chapter is a vital step in the work of the wider thesis, as it defines a new architecture for structural design software on which the rest of the work of the thesis relies. This architecture is itself a highly original contribution to the field of computational design, first outlining features which design software should have and then specifying how to best realize these features.

3.2 Models

In order to describe a design tool which will facilitate multiple modeling methods for a single structure, one should first define what a "model" is. A model is an abstraction of the real world, a "substitute for a real system" to be used "when it is easier to work with a substitute than with the real system" (Ford, 2009). A model is used to produce some output, typically a response of the system to a stimulus. An artifact is a man-made object, and the term is used here to establish a clear distinction between the real world structure (artifact) and a model of it.

In the domain of structural engineering, this distinction between model and artifact is sometimes inappropriately blurred. Designers of artifacts should take care to critically interpret the output from a calculation or modeling process in the understanding of what a model actually is. Although providing useful information about how a real system behaves in a given setting, the model must be assessed as much on its formulation as on its results. No model captures entirely or with full accuracy the behavior of the real system which it abstracts, and a thorough understanding of the omissions and simplifications made in abstracting the real system are a prerequisite for a sound understanding of the predictions made.

Many models of a single system can exist, none of which change the underlying physical reality of the system. Whether we choose to model an airplane in flight as a concentrated point mass or as a complex aggregation of aircraft components, people and luggage distributed over a large three-dimensional space has no effect on what the airplane really is, or on how it will really behave. Our choice of model will, naturally, affect the predictions it makes, and an understanding of the limitations of the model is essential in reliably interpreting these predictions.

There are many ways to distinguish models, and one that will be used in this chapter is the degree of resolution, which is the degree of detail of the real system which the model captures. High resolution models are relatively faithful to small details of reality, capturing nuances and subtleties omitted by low resolution models for the sake of simplicity. The resolution of a physical model is largely independent of how it represents the governing physical laws of the system. Certain assumptions are made about physical behavior, and various assumptions differ to varying degrees from reality. As an example, linear behavior (where a change in a parameter of the model produces a proportional change in a response) may be assumed in scenarios where the natural phenomena is really nonlinear, but close enough to linear that reliable predictions can be made. Simply increasing the resolution of the model does not a priori make it any more or less likely to capture this nonlinear behavior. This is the type of pitfall that designers using sophisticated modeling software may be prone to. It is tempting to think that by continually increasing the resolution of a model, one can begin to converge on the behavior of the real system. This neglects to consider the way in which the physical phenomena governing the behavior of and the interactions between the components of the model are represented. It could be argued that an increasingly accurate representation of these phenomena is itself a type of increase in resolution, but in this thesis the detail of representation of components and of the physical phenomena which govern their behavior will be considered distinctly.

3.3 Models in Structural Design

The process of structural design should not develop a dependence on a single modeling method, as different methods offer different benefits and limitations. Long before the advent of the computer, simple low resolution models which could be built and understood by a single designer were used to understand and predict the behavior of structural artifacts. (Akira, 1999). The physical models built by famous twentieth-century engineers such as Pier Luigi Nervi and Heinz Isler (Chilton, 2000) are examples of non-computer models that have been used to great effect in the design of efficient, elegant structures.

Such easily understood models bring the designer closer to an intuitive understanding of physical behavior, of the effects of design decisions, and of the mechanism by which these effects are linked to the designer's decisions. A clearly understood model is a transparent model, giving the designer a heightened insight into the mechanics of the system. Highly complex models, requiring the processing power and memory capabilities of a computer, remove the designer from this intuitive understanding. They can be thought of as opaque, or 'black box' models. Little is understood by the user about precisely how responses derive from input, and design becomes little more than a blind trial and error process.

Despite such shortcomings, it should be clearly stated at the outset that what is *not* being advocated here is the abandonment of complex computer modeling methods in the field of structural design. What is advocated is a rethinking of the role such methods play and of their place in the overall design process. Relatively low-resolution methods of modeling structural behavior, such as the strut and tie method (Schlaich et al., 1987), graphic statics (Zalewski et al., 2009), or the idealization of a tall building as a cantilevered bending beam are useful in quickly establishing a first-order behavioral model of the structure being designed. (There are classes of structures for which simplified models provide an accurate representation of reality but, for the purposes of this discussion, their ability to approximate the behavior of a much more complex system is of greater importance.)

The ease of manipulation and of response generation when using such models makes it possible for the designer to explore the design space quite quickly. Intuitive, low resolution models free the designer from the multiple small details and allow him to direct his design towards favorable areas of the design space. The optimization aspects of the software, introduced in chapter 4, also benefit from the ability to use low resolution models during the initial design phase. Optimization problems using these models

can be several orders of magnitude smaller than an optimization problem based on a highly detailed FEM model of a structure. The problems are easier to work with, and run times are dramatically reduced (Afonso et al., 2010).

Low resolution models miss much of the detail of a real system, a limitation that varies in importance depending on the model, on the system and on the desired response. This limitation can be overcome later in the design process - once a broad exploration has occurred and the general characteristics of a solution have been defined - by modeling the system using more sophisticated high resolution techniques which capture a desired degree of the missed detail. In practice, this is often what happens. A designer creates a simple model of the real system, often sketched by hand on a sheet of paper (or any available writing surface), and uses this model to gain a first-order understanding of structural behavior. Once this high-level behavior is understood and a conceptual design exists, a detailed finite element model is built, giving a much more thorough prediction of the behavior.

A distinction is drawn here between two types of models in the field of engineering. The first of these, a *structural model*, is defined as a physical abstraction or idealization of the real physical system (which, for the remainder of the chapter, will always be a structural artifact). The structural model describes the artifact as an aggregation of some understood and analyzable components, such as beams, plates and springs. A grid-like structural artifact could, for example, be modeled as a pin-jointed truss or as a rigid-jointed frame. The same artifact could also be modeled as a series of finite elements.

The second type of model is a *mathematical model*. This is defined here to be the mathematical abstraction of the structural model, used to analyze the model and determine responses. It consists of the mathematical constructs (equations, matrices, etc.) that describe the structural model, and the methods to manipulate and solve these constructs in order to predict the artifact's behavior. Just as numerous structural models can describe a single artifact, so too can numerous mathematical models describe and analyze a structural model. For example, the same truss model of an artifact could be solved using small displacement or large displacement theory, both of which require a different set of equations and can have different solution methods, and as such constitute distinct mathematical models.

The progression from artifact to structural model, as from structural model to mathematical model, is considered an increase in abstractness of representation.

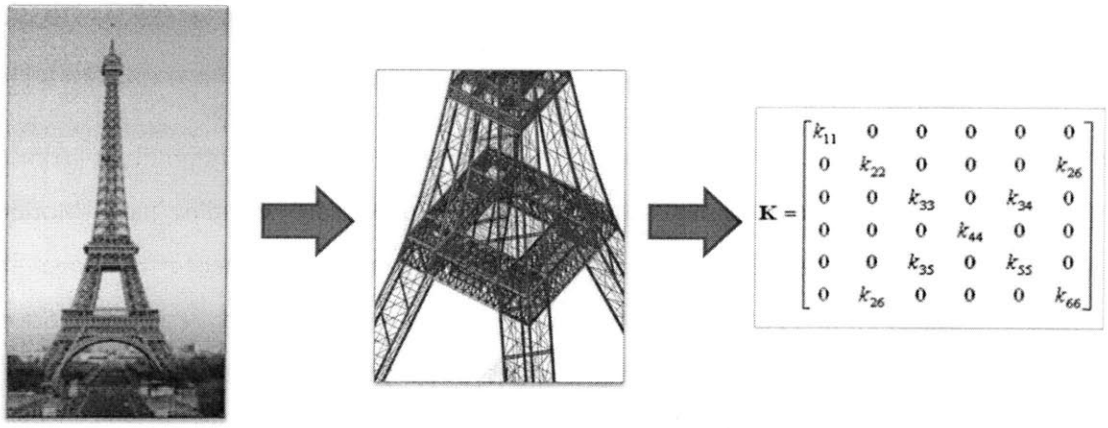


Figure 3.1 Increasing abstraction in the progression from artifact to mathematical model

Figure 3.2 shows this progression from an artifact to a structural model on a UML (Unified Modeling Language) entity relationship diagram. Each type of model can be thought of, and will be referred to, as an abstraction of the entity to its left in this figure. The significance of the notation at the end of the entity connectors, representing cardinality, is discussed in Section 3.5. For a further discussion of UML, which is used extensively in this chapter to describe the architecture of the developed software framework, see Larman (2004).

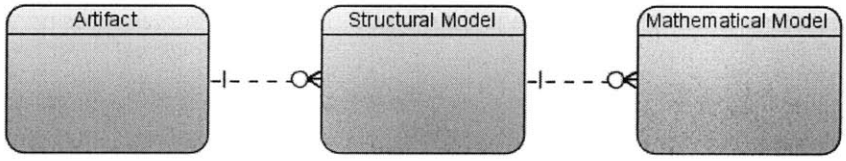


Figure 3.2 UML entity relationship diagram of the relationship in Figure 3.1

This particular approach of using multiple modeling methods in computational design, and the arrangement of models in increasing levels of abstraction, is an original contribution of this work. Researchers have previously incorporated some aspects of this approach in their work (Mackie, 2001a; Patzak and Bittnar, 2001), but the current approach to structural software with a particular emphasis on design is unique.

3.4 Example – Motion-Based Design of a Tall Building

In order to demonstrate the utility and advantages of multi-model representation of an artifact during the design process, this section examines the design of a multi-storey building using Motion-Based Design (MBD) (Connor, 2003). The essence of this approach stands in contrast to traditional strength-based design methods, in which a structure is designed to be strong enough to resist the most extreme loads to which it will be subjected in its lifetime. Under the strength-based approach, prescribed bounds on the allowable motion (displacement, velocity and acceleration) of the structure, which must be adhered to for occupant comfort and for negligibility of structural damage levels, are checked after a design has been determined on the basis of strength. In the case of tall buildings, the allowable motion limits under lateral (earthquake and wind) loading are usually exceeded, requiring iteration of the design process and, typically, heuristic modifications to reduce motions.

In motion-based design, the opposite approach is taken. The structure is designed to stay within motion limits, and the strength is checked afterwards. For structures which are susceptible to lateral loading, it has been shown that the strength constraint will often be satisfied, and no further iteration will be required. The motion-based design of tall buildings, therefore, requires the designer to consider the dynamic characteristics of the structure at a very early stage in the design process. This motivates the use of simplified models, in which an understanding of dynamic behavior can be intuited and used in the initial design phase to a much greater extent than could be feasibly achieved using a more complicated high resolution model.

In working to meet the constraints on dynamic motions, the best distributions of mass, stiffness and damping throughout the building are by no means obvious, and designs may behave very differently and unexpectedly under various dynamic excitations. As a result of such complications, it is useful to model the building in as straightforward a way as possible during the initial stages of motion-based design, so that a wide range of designs can be conveniently explored. The number of design variables is thus reduced, and the effects of design changes are easier to understand.

A single change in a simplified model may well be representative of an order of magnitude more changes in a more complicated model. Take, for example, the modification of the inter storey height. In a highly representative, high-resolution model of the building, this would at the very least involve a change in the length of every structural member linking the two floors, and potentially many more

operations. In a simple model it entails fewer, simpler operations. During the initial design phases, when the designer should seek to explore the design space broadly, the task of design exploration becomes much less onerous if the model being manipulated is of low resolution. If optimization algorithms were to be applied in the MBD process, the availability of these low resolution models at the initial design stage would greatly uncomplicate and simplify the optimization process.

The first such simplified model typically used in the MBD literature takes the form of a *'stick model'*, where the mass (and the rotational mass) of the building is discretely lumped at floor levels, and the action of the building elements at each floor level in resisting lateral loading is represented by massless beam segments which link the lumped masses.

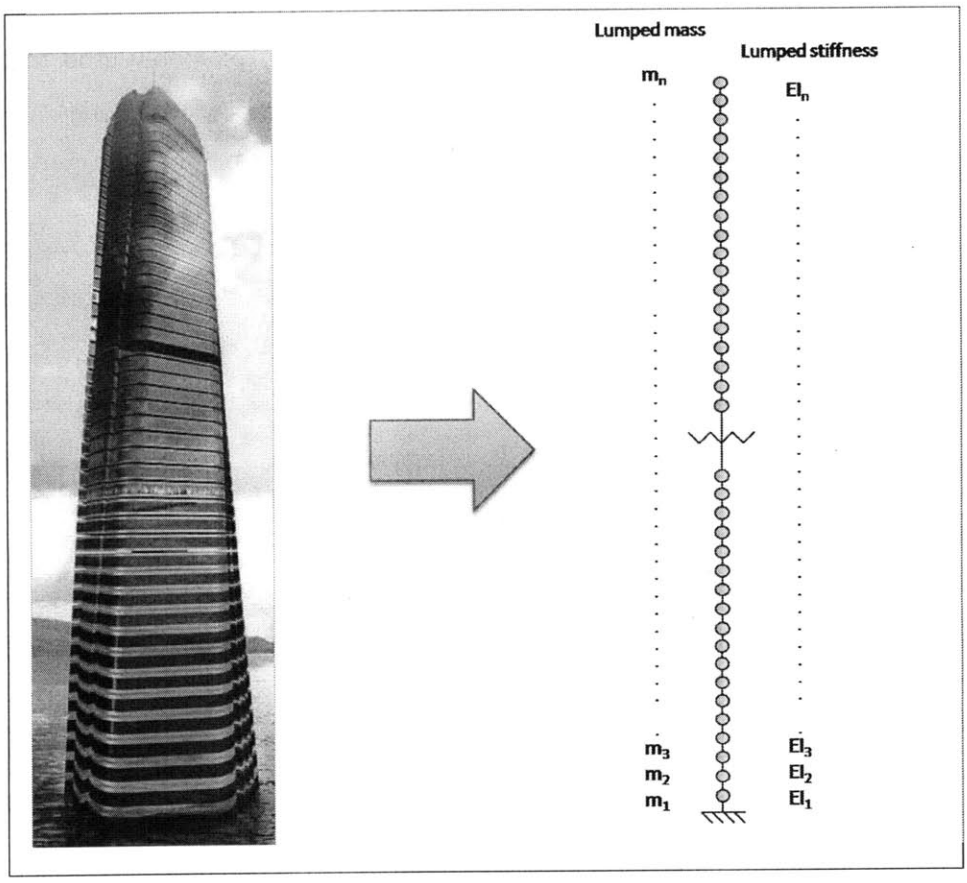


Figure 3.3 A tall building, and a potential (simplified) stick model representation

This easily modified and analyzed model could be used to determine a favorable lateral stiffness for the building, subject to various excitations. A certain frequency for one of the dominant modes of oscillation could be targeted, for example. Changes in the geometry of the stick model could update certain geometrical information about the artifact itself, such as inter-storey heights. Under some broad

assumptions, other design changes in the stick model could also be reflected in the computer's representation of the artifact. For example, the cross-sectional area of columns at a certain floor level could be increased in proportion to shear stiffness of the relevant discrete beam element.

There are various mathematical techniques that could be used to 'solve' the stick model, each of them yielding varied responses. A static model is straightforward, showing the axial load carried in all the columns at a certain floor level under gravity. A quasi-static analysis of lateral load could easily be used in this context to approximate wind and seismic effects. Mathematical models which capture the dynamic effects are of greater use here. They could be simple eigenvalue solvers which determine modal frequencies and mode shapes for various modes of oscillation, or they could be more sophisticated solvers capable of simulating a time history of the dynamic response of the model to any dynamic excitation. (Mathworks' Simulink® is a software package commonly used for the complex solution of stick models in this way). All of these mathematical models abstract and analyze the same structural model. This is shown graphically in the figure below.

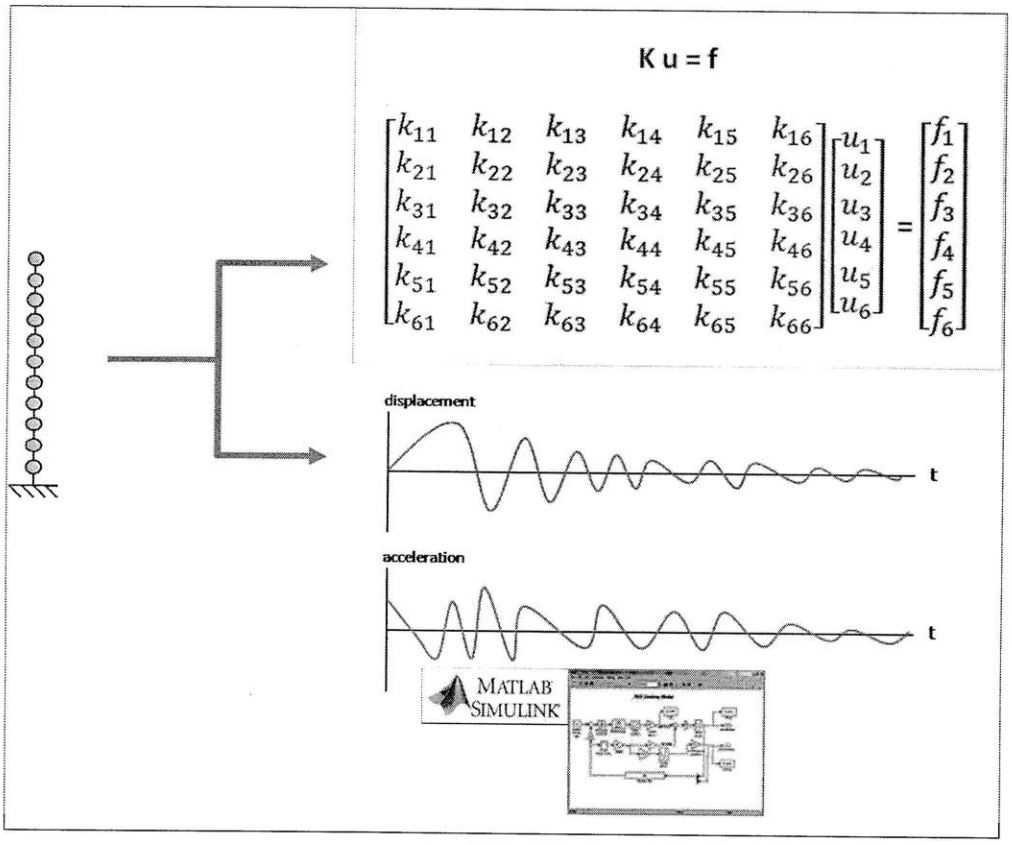


Figure 3.4 Mathematical models used to abstract and analyze the stick model (notional representations only – not actual models of system)

Another simple structural model which approximates stiffness and mass distributions is a cantilever. Unlike the previous lumped-mass shear beam example, the cantilever approximates the mass of the building as continuously distributed. Analytical solutions for static and dynamic behavior are well documented, meaning that an analytical expression for these distributions could be arrived at quickly.

Once a favorable lateral stiffness distribution has been defined, the building is next modeled as a frame or as a pin-jointed truss, capturing behavior at the individual member level rather than at the overall floor level. Again, some broad assumptions have to be made in order to translate the properties of the stick model, such as mass and stiffness distributions, to the frame model. This translation from stick model to frame, via the information known about the artifact or directly, is either hardcoded under a set of assumptions, or left to the user to specify.

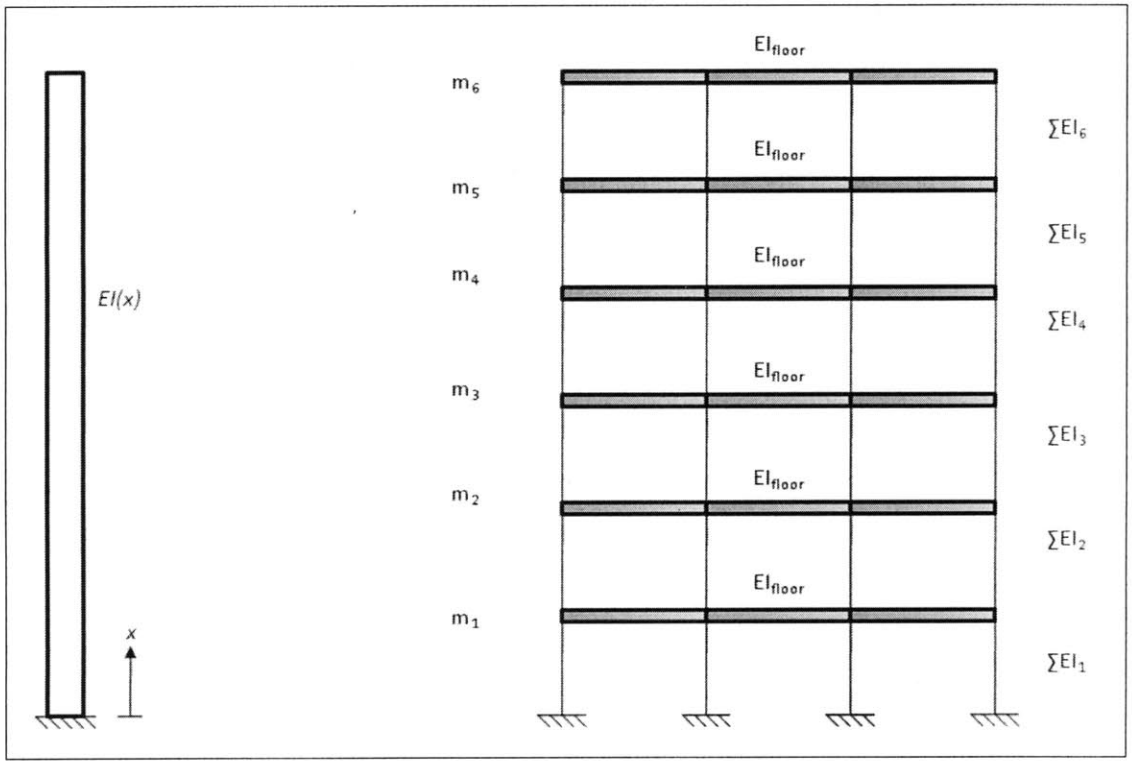


Figure 3.5 Structural models of a building: Cantilever and frame

The frame model is be used to make design changes that affect the characteristic shape of the building, and to determine stress and strain of individual elements. Parameters such as weight and cost of the artifact can now be estimated using this more sophisticated modeling method, and architectural

concerns can be reconciled with the desired stiffness and mass distributions determined from the stick model. Note that distinct mathematical models can be used to analyze the structural frame model. These could be stiffness methods, flexibility methods, small or large displacement methods, etc., and could incorporate dynamic or exclusively static analyses. If dynamic effects cannot be captured, the stick model will automatically update to reflect the properties of the frame model, and so this can be analyzed to estimate dynamic performance instead.

The final increase in resolution is to use an FEM solver for the building, in the confidence that all the design decisions made on the basis of the simpler models have led the designer to a good region of the design space, having addressed dynamic and architectural concerns. Many costly (in terms of time and computational effort) FEM dynamic analyses will not have to be run iteratively in order to determine a good motion-based design for the building. Since a good design will already have been arrived at using lower-resolution models with lower computational demand, FEM analyses serve primarily to verify the determined design.

All these structural models are tied to a single artifact, so changes to any of the models can be used at any time to indirectly modify the artifact. Via the artifact, changes in one model can propagate to another. The entire set of models and the representation of the artifact, as described up to this point, can be seen in Figure 3.10 at the end of the chapter. Before this presentation, the details of the framework which link the models is developed in section 3.5.

As well as providing an illustration of the benefits of using the software design strategy specified in the wider chapter, this section is itself an original contribution to the field of motion-based design, outlining how the MBD approach could be integrated into computational design.

3.5 Object Oriented Programming of Multiple Models

The computer is now a ubiquitous tool in engineering and architecture. From the first stages of study through to the highest level of professionalism, software packages have a firmly established place in the design and analysis processes. In order to bring the full power of the use of multiple modeling methods to bear on the design process, a software architecture which allows these models to integrate as seamlessly as possible in an overall framework must be developed. The purpose of this section is to

provide a technical description of this framework, to the extent that a reader with programming experience could develop design software based on the approach.

The logical structure of OOP makes it an obvious paradigm to use in building such a framework. This stage of theoretical and software development is influenced by the knowledge that a strong element of structural optimization will be later included, as described in chapter 4. The framework will be extended to account for the inclusion of optimization, but all aspects of it described in this chapter will remain unchanged.

This distinction between structural and mathematical models is an implementation of the *Model-Analysis Separation* design pattern described in Heng and Mackie (2009). The separation of structural models from artifacts is somewhat more novel.

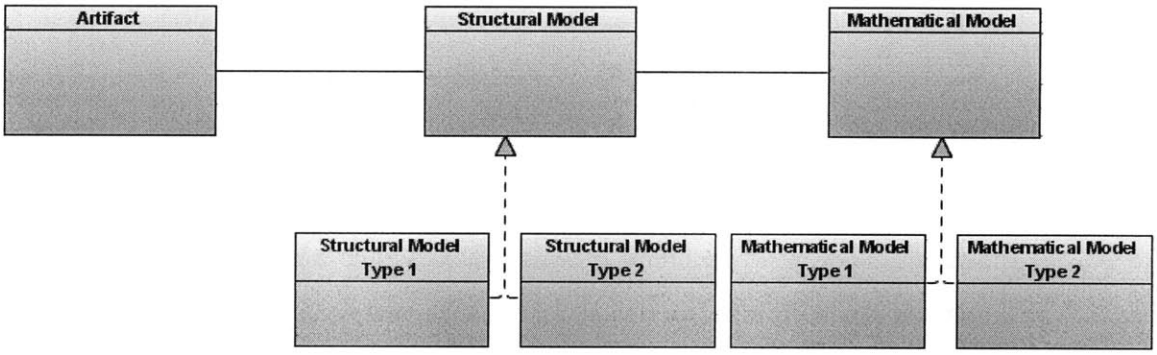
The OOP concept of inheritance is used extensively in the design of this software framework. Beyond the immediate benefit of logical organization, the use of inheritance means that communication between the artifact, a given structural model of the artifact, and a given mathematical model of the structural model can occur at the parent class level. This is crucial to the extensibility of any program developed using this framework, and will be revisited.

Figure 3.6 (a) shows a generic representation of where links would be specified in an inheritance hierarchy. A new object for mathematical modeling, for instance, needs only to know how to build itself from a structural model, and how to analyze itself. This figure shows the essential characteristic of Model-Analysis separation design pattern, embodied in the distinction between structural and mathematical model objects, combined with the separation of structural model objects from the artifact object. This is, it should be noted, an extremely simplified view of the actual framework, included to emphasize the separation between these three ways of considering a structural system – an actual artifact, a structural model of that artifact, and a mathematical description of that structural model. These can be thought of as three distinct views of the system.

The artifact object is essentially a data structure which encapsulates a physical description of a structure, but not necessarily in any way that could be directly analyzed. It should be thought of as a blueprint; an aggregation of information from which one could actually build structure in reality. The artifact object in the code contains information on the geometry of the structure, the material properties, even the thickness of members in specific locations. The more detailed the information contained in the artifact object, the higher the resolution of the models that can be abstracted from it.

There is nothing in the rules of this framework to enforce the existence of an underlying artifact for every model. Engineers are comfortable working with models without thinking too much about what they really represent. The key insight is that any model object *can* have an underlying artifact object, and that the framework allows this to occur. The existence of an underlying artifact is important if many modeling methods are to be used, in order for them to have a common point of reference. The three representations of a structure in this software framework – the artifact object, a structural model object and a mathematical object – can be thought of as distinct *views* of a structure. They each represent the structure in increasingly abstract ways, yet are all ways of displaying the same thing.

Figure 3.6 (b) shows the status quo approach of linking modeling components directly, without any of the hierarchical or abstraction features of the new approach.



a) Links between model and artifact objects occurring at a high hierarchical level

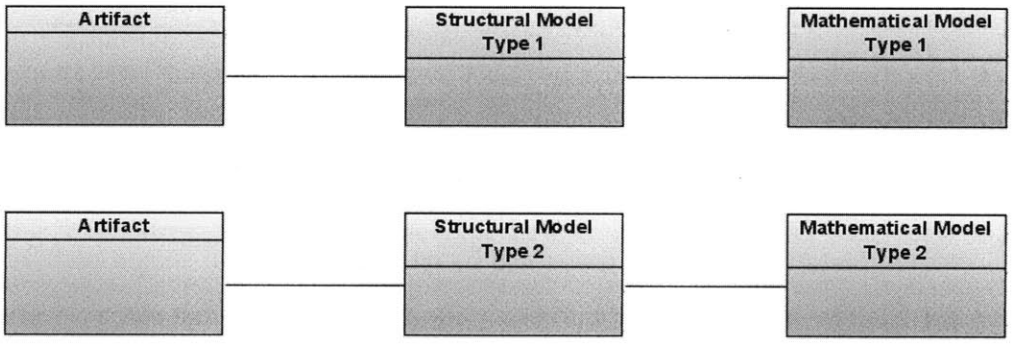


Figure 3.6 b) Status quo approach of links between individual modeling classes. No hierarchical abstraction

In order to benefit fully from the object-oriented approach, a further subdivision of the different modeling techniques, capturing the commonalities of the models in each subdivision, is required. The logic behind this approach is that similar types of models will have similar ways of representing and operating on data. In order to enable the kinds of code-sharing benefits, extensibility and flexibility associated with the inheritance and polymorphism features of OOP, objects which inherit from a common parent should have as much in common as possible (Booch, 2007). To achieve this, a classification of structural and mathematical models by type is necessary.

3.5.1 Structural Models

Structural models are classified as either continuous or discrete. This distinction is, of course, philosophically debatable, since a digital computer will ultimately represent a continuum discretely. This discussion will not be pursued. In the framework, continuous structural models are those which represent the structure as a continuum before any decision is made on how to analyze them mathematically. Finite element models are the most obvious member of this class of model. They treat the structure as a continuum, and discretize it for the purposes of analysis in a way that is not necessarily tied to any notion of how the structure is composed of individual components. Members which would be considered discrete entities in the building industry, such as beams or columns, are discretized in any number of ways, none of which necessarily bear any discernible physical relation to the member's composition.

Discrete structural models represent the structure as an aggregation of discrete elements. Truss models and frames are two such examples, which are then further subclassified by their dimensionality. All of these model classes ultimately inherit from a 'structural model' parent class, which captures the features shared by all structural models. Intermediate levels in the inheritance hierarchy gather common features of models in each subdivision. The structural model classes contain no analysis methods; they contain only data and methods necessary to abstract the reality of an artifact. Any structural model type which is later created can inherit from the most suitable intermediate parent class, or, if none is deemed suitable enough, from the structural model parent class.

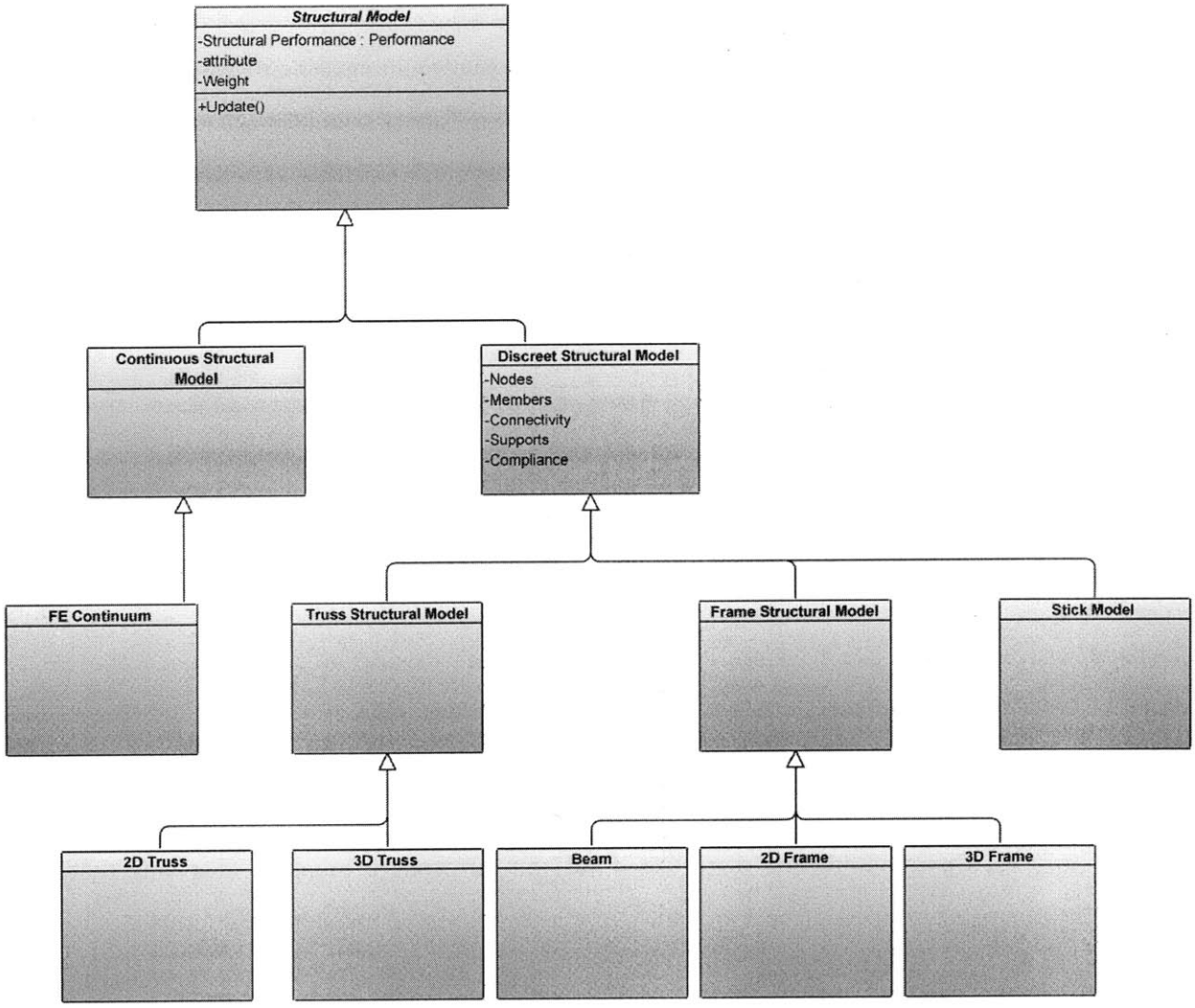


Figure 3.7 Hierarchical class diagram of structural models

3.5.2 Mathematical Models

Mathematical models are also classified as continuous or discrete. Continuous mathematical models are not simply the mathematical model associated with a continuous structural model. Rather, they are models which themselves are continuous in nature, such as the use of continuous differential equations to mathematically model a structural model. Discrete models are subdivided into those that use a stiffness matrix, and those that do not. There are limitations on the types of mathematical models that can be used to model a particular type of structural model. This is seen in the positioning of associative

links between mathematical and structural model classes in the complete framework class diagram, later shown in Figure 4.7.

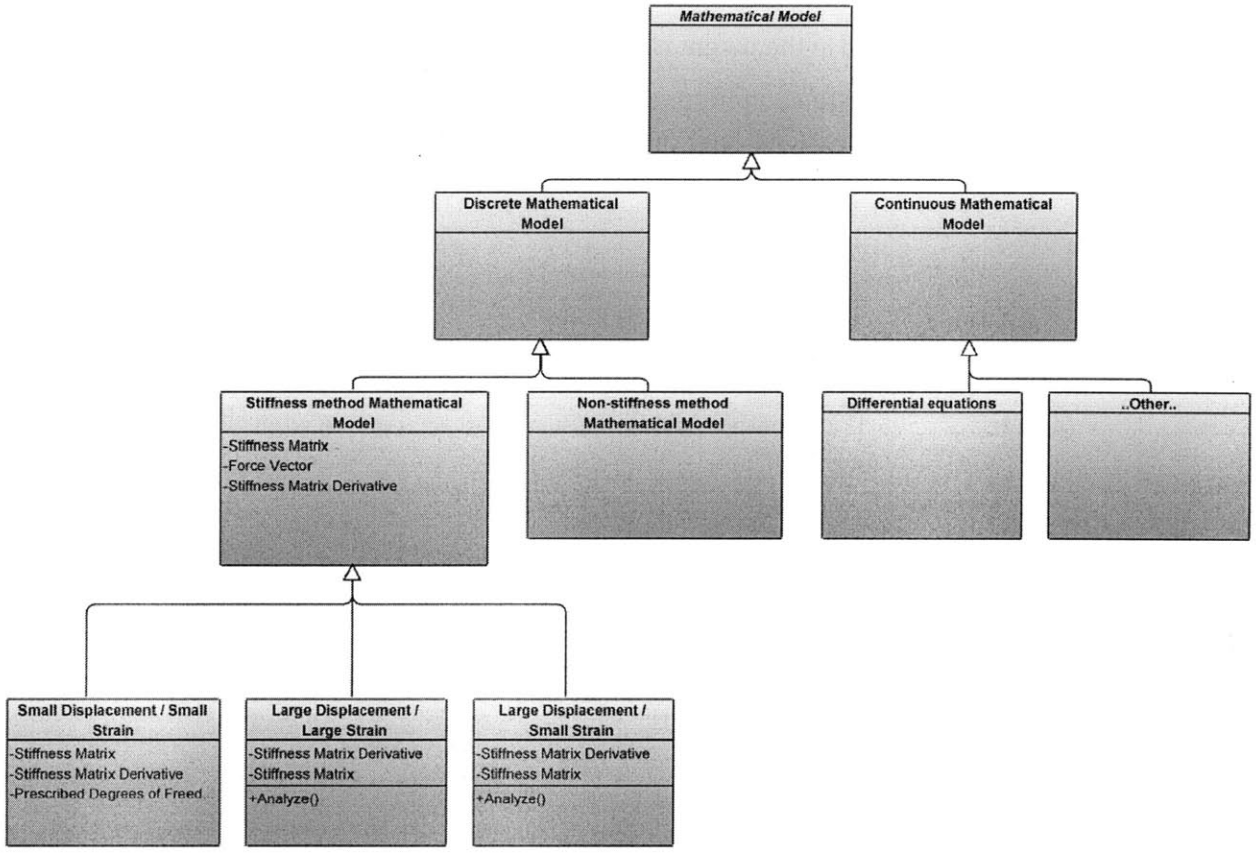


Figure 3.8 Hierarchical class diagram of mathematical models

3.5.3 Extensibility

All the classes which are not at the lowest level of their branch in the hierarchical tree are abstract. An object cannot be instantiated from them, and must instead be instantiated from a child class that lies at the bottom of their particular branch of the tree. The abstract parent classes contain code common to all classes that inherit from it, and also serve to manage the links and communication between the different model objects and the artifact object.

Interfaces, contracts specifying methods that any class which implements them must include (see Booch [2007]), are defined for each of the parent classes at all levels in the hierarchies. In order to extend the

framework to include new mathematical or structural modeling methods, the developer decides which parent class is most appropriate for his intended model class. The new class inherits the appropriate parent class and, by default, the appropriate interface. This gives the new class access to data fields, structures and methods specified in the parent class. It specifies via the interface the methods which the developer must provide in order to conform to the framework. By convention, the names of interfaces begin with the letter i in order to distinguish them from classes.

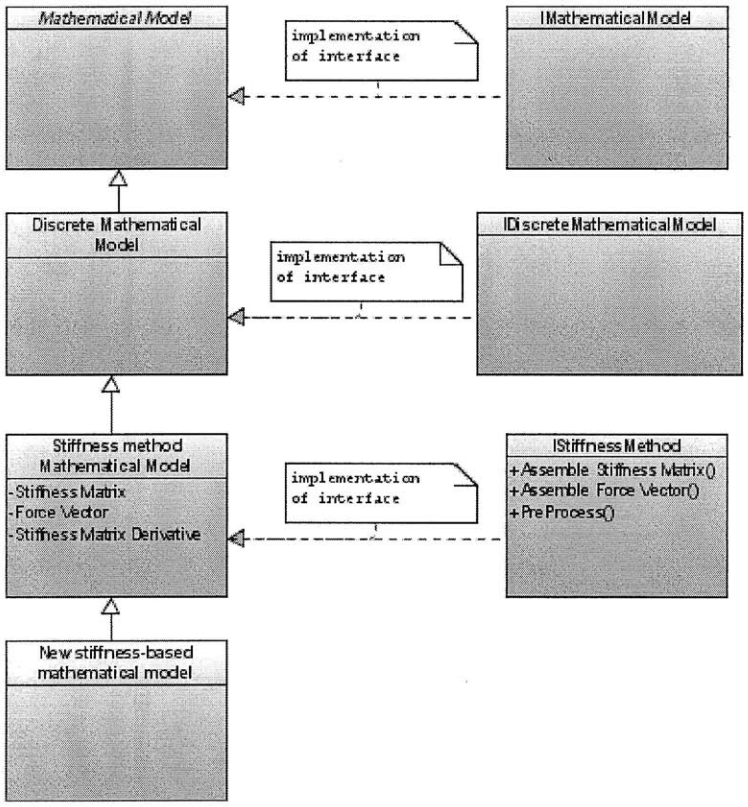


Figure 3.9 Inheritance and interface implementation

For instance, a class which inherits from the *StiffnessMethodMathematicalModel* class is obliged to implement the *IStiffnessMethod* interface. Any developer introducing a new stiffness method-based mathematical model is automatically provided with the blank data fields to store information about the stiffness matrix, force vector and displacement vectors. The *IStiffnessMethod* interface forces him to provide methods to assemble the force vector and stiffness matrix from the structural model, and the *IMathematicalModel* interface forces him to provide a method to analyze the model and distribute the

analysis results to the relevant points in the structural model. This is shown in Figure 3.9. The entire framework, showing all previously described classes and how they interact, is shown in Figure 4.7 at the end of chapter 4.

The use of various geometric modelers is allowed by this framework. These are programs which allow users to rapidly and flexibly generate geometric representations of designs, and are often used by architects as explorative tools. *Rhinoceros®* (McNeel, 2007) and *CATIA®* (Dassault Systemes, 2002) are examples of such explorative tools. In the described framework, these modelers would logically interact with the structural model and artifact objects. A modeler used in this context must specify methods to modify and render the data held in these objects, and the developer of such a modeler must fully understand the data representation of these objects. Despite the restriction of having to know the exact data structures, the use of a framework such as this allows for the integration of data modeling software with optimization algorithms and analysis engines, a feat which is typically achieved using scripting languages and a once-off ad-hoc approach.

The provision of real-time or near real-time analysis, which enables the pursuit of a responsive simulation of a structure as well as a variety of form-finding methods, is naturally managed using events and event handling in OOP. The event features of an object oriented language such as C# or Java make them natural candidates for developing a program which can provide real-time feedback to users and can automatically return function evaluations to optimization algorithms.

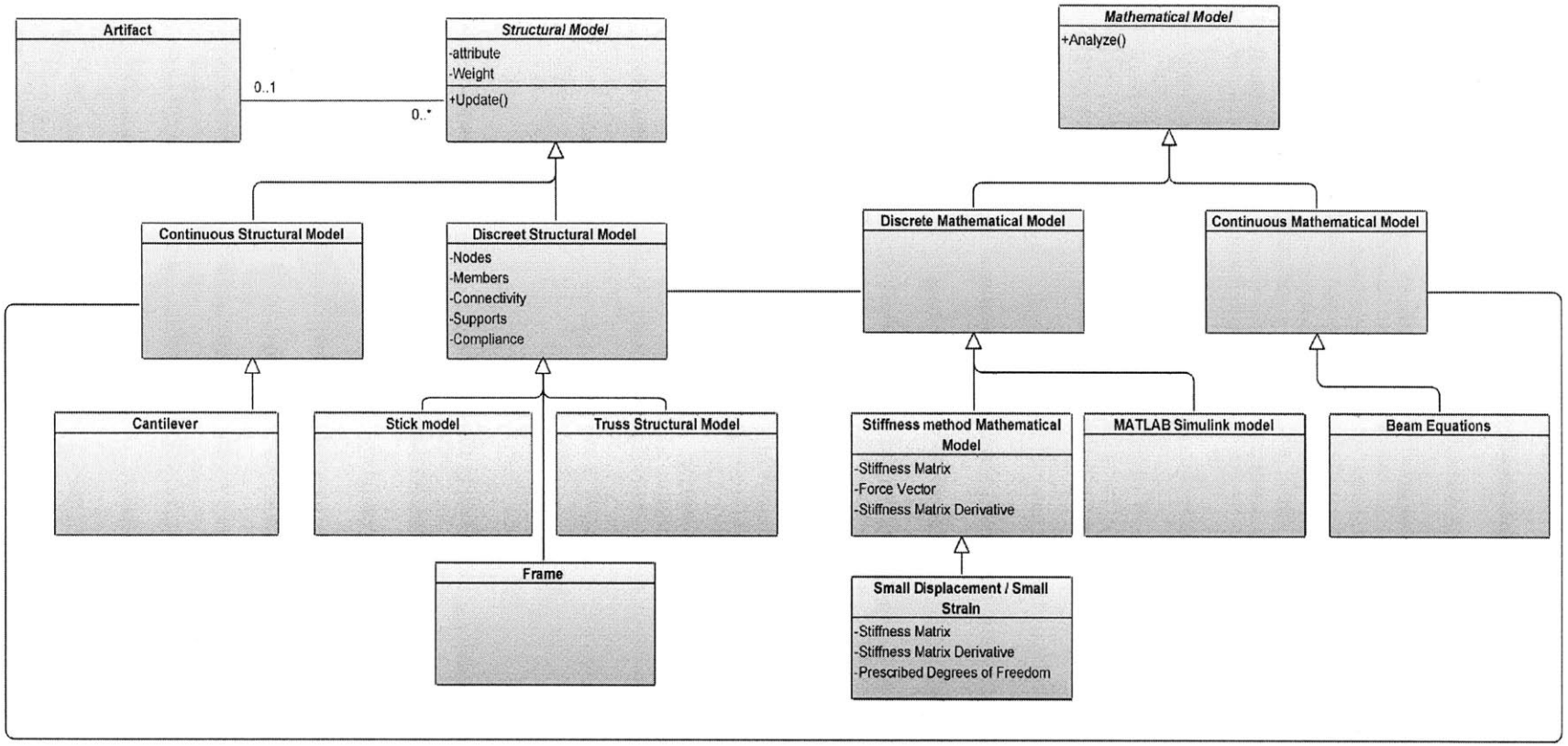


Figure 3.10 The multiple model approach to MBD, using the developed framework

3.6 Summary

The goal of this chapter was to develop and describe the framework for a structural design program which enables a software developer to meet the needs identified in chapters 1 and 2 by developing a structural program with a strong design sensibility which can be built upon and extended in a logical predefined way. The use of multiple modeling methods and the clear separation of all levels of abstraction in the design of the software framework are an original way to develop software which makes use of the strengths of various modeling methods in their best context. This enables the development of software in which intuitive models which give a strong conceptual design sensibility can be seamlessly integrated with higher resolution models for detailed design and verification. The framework provides a clear means to implement real-time interactive software and to allow geometric modeling software to integrate with analysis components. Developers can write code which, if it respects interfaces defined by the framework, can easily integrate with the existing components.

The framework is a means to meet these identified needs, and is a highly original contribution to the field of structural software. It is the foundation for much of the rest of the thesis. The extension to optimization, later described in chapter 4, is built on this framework, and the conceptual design program described in chapter 5 is developed on its basis.

Chapter 4 Extension to Optimization

4.1 Introduction

The goal of this chapter is to include optimization in a program for use at the conceptual design stage. In doing so, the work aims to make many existing optimization methods more immediately useful to designers and to further encourage creative exploration of design possibilities.

The deficiencies in existing optimization methods as applied to conceptual structural design outlined in chapter 2 are addressed by identifying and describing key improvements to be made. The software framework is extended to incorporate existing optimization algorithms and these improvements, and a description is given of the optimization components of an implemented program which is developed based on the approach of this thesis.

4.2 Areas for Improvement in Structural Optimization

A structural optimization problem seeks the values of a set of design variables x_1, x_2, ..., x_n, aggregated into a design vector $\mathbf{x}$, which minimize the value of some objective function $f(\mathbf{x})$. The objective function in structural optimization is usually built from parameters which evaluate some measure of the structure's performance, henceforth referred to as performance parameters. Structural optimization problems almost always belong to the class of constrained optimization problems, where a number of inequality constraints and equality constraints (defined by functions $g_i(\mathbf{x})$ and $h_j(\mathbf{x})$) must be satisfied. The design vector $\mathbf{x}$ is usually bounded above and below by a corresponding vectors of upper and lower bounds, $\mathbf{x}_{lb}$ and $\mathbf{x}_{ub}$, typically to ensure that results make physical sense (Fox, 1971).

$$\min_{\boldsymbol{x}} f(\boldsymbol{x})$$

subject to $\quad g_i(\boldsymbol{x}) \leq 0 \quad i = 1 \ldots n_{ic}$

$$h_j(\boldsymbol{x}) = 0 \quad j = 1 \ldots n_{ec}$$

$$\boldsymbol{x}_{lb} \leq \boldsymbol{x} \leq \boldsymbol{x}_{ub}$$

Despite considerable investigation in academia and occasional application in industry, structural optimization has been limited in structural design by a number of factors. The purpose of this work is to identify a manageable subset of these factors, and to address them in order to make structural optimization more applicable to the structural design process in general.

4.2.1 Handling of Fuzzy Problem Statements

One of these limits on structural optimization is the inherent difficulty in taking the often ill-defined notions in a designer's mind about what exactly constitutes a good design for the situation at hand, and using them to define a precise problem statement. For a given design problem, the makeup of the objective and constraint functions, and even the content of the design vector, is highly subjective, varying from designer to designer.

As an example, consider the quantification of a designer's relative preferences for different objectives in a multi-objective problem. A number of strategies to deal with multi-objective problems in structural optimization exist (Coello Coello, 1999). One of these, the weighted sum approach, involves the expression of each objective in a separate objective function $f_i(\mathbf{x})$, and the linear combination of these functions into a single objective function, given by

$$min \sum_{i=1}^{k} w_i f_i(\boldsymbol{x})$$

where w_i is the weighting of each function, in proportion to its perceived importance in the resulting design. Since virtually all structural design problems are multi-objective, a major challenge for any designer using multi-objective techniques lies in deciding which parameters to include in the objective

function, and the relative importance of each. The range of possibilities for these choices corresponds to part of the fuzziness of the structural optimization problem.

Much of the problem statement depends on the designer's opinion, which can vary substantially with time and context. Such uncertainties in a given designer's problem statement and the subjectivity in problem statements in general introduce a large amount of fuzziness to the structural optimization problem, which is recognized and addressed to varying degrees by researchers and developers of optimization software. Success has been achieved using fuzzy logic and fuzzy programming (Rao, 1987), which deals with the uncertainty at the input stage and delivers a single solution. Other work has focused on using stochastic methods to present the user with a wide range of solutions from which to choose (Von Buelow, 2008), often in combination with constraining the shape to lie within a spatial envelope (Sassone, 2008), thereby dealing with the fuzziness by allowing the user to manually post process a range of results within certain bounds which vary with the degree of fuzziness of the problem.

While this fuzziness of the problem statement exists to an arguable degree when considering performance parameters which are naturally expressed numerically, such as weight, stiffness or cost, its presence is undeniable when considering architectural and aesthetic evaluation criteria. Such criteria are difficult enough to describe qualitatively, let alone express mathematically in a way that an optimization algorithm can work with.

It becomes evident from considering such multi-objective and fuzzy optimization literature that it is beneficial, in terms of encouraging creativity in design and thorough exploration of design possibilities, to allow users to interact with structural optimization software in ways that allow them to specify preferences driven by personal and subjective judgment. Although some researchers have captured aesthetic concepts analytically using shape grammars (Shea and Cagan, 1999), the best way to capture these will vary for each designer. This approach is not pursued here, since a customization to suit individual users' aesthetic preferences would involve reprogramming and redevelopment of tools, which limits the appeal of the process for those that do not have experience in developing or customizing code.

Recognizing the need to make structural optimization methods more useful to designers with little formal optimization training, this motivates the use of structural optimization in a flexible manner, where the problem statement can be conveniently manipulated by the designer in an intuitive graphical environment. Designers can explore the effects of varying problem statements, such as the changing of

relative importance in a multi objective scenario or the relaxation of certain constraints, without the need to carry out any programming. Parameters which are difficult to quantify and involve some fuzzy ambiguity, such as aesthetics or practicality, are left entirely to the designer's judgment in this approach.

Although work has been done on capturing designer preference in various ways, a simple tool that would allow designers to conveniently experiment with modifying the problem would encourage creativity in structural design using optimization, and would better enable those with no optimization training to understand the significance and effect of varying problem statements. While it may not be the best way to mathematically approach the problems, the sacrifices made go a long way towards making the software more practically usable.

4.2.2 Handing Control Back to the Designer

Another fundamental limitation identified by designers who have not been formally trained in the field of structural optimization is the relinquishment of control to the optimization algorithm which is enforced by most optimization tools, as observed by the author in conversations with such designers. The user defines a problem statement, runs the optimization algorithm, and is presented with results. This can lead to the user feeling distanced and even alienated from the design process, deprived of control over design decisions.

The interactive optimization tool envisaged in the previous section would give designers as much control as desired, by allowing them to constrain the optimization problem statement as much as desired. This controls the balance between optimization-driven and user-driven design decisions.

The tradeoff in handing control to the designer instead of the optimization algorithm is the loss of optimality. As the optimization problem becomes more constrained by the designer, the optimum value of the objective function necessarily either gets worse or stays the same – it cannot improve. The philosophy of this approach involves handing more control to the designer so that optimization is used as a search tool for optimally-directed solutions rather than as a standalone way to produce optimal designs.

4.2.3 Facilitating Extensibility

There are many instances in the structural optimization literature of researchers developing software by using a scripting environment to stitch together optimization algorithm, analysis software and user-oriented features such as rendering and interface functionality. The method of integration of the optimization algorithm depends entirely on the developer's approach, and is often so opaque that the algorithm cannot be conveniently replaced by another developer. While this would at first glance appear to be a software implementation issue, its limiting effect on academia is not trivial.

In a field where the performance of algorithms across different types of problem varies greatly, and where there is often debate about which type of algorithm works best in a given setting, it is desirable to have a standard framework which allows algorithms to be removed and replaced as a discrete component of software. In a scenario where the research community adopted such a framework in place of custom approaches, different optimization algorithms could conveniently be applied to a problem by a developer other than the one that initially developed the software. Results could more easily be verified and reproduced, and sharing of completed work among the research community would likely increase. The work of researchers in the field of structural optimization could more immediately become useful in a real-world design tool, with other researchers working simultaneously on the modeling, analysis and user interface aspects of the tool.

4.3 Optimization in the Software Framework

The logical layout of the design software framework described in chapter 3 facilitates the developed program's extension to structural optimization, and forms the basis for an interactive design tool incorporating structural optimization. The underlying philosophy governing this extension is that the optimization algorithm is treated as a user of the system, just as the designer is. The algorithm, via an *optimization object* (described in section 4.3.1), modifies the structural model, which triggers an automatic analysis by the mathematical model. Analysis results are automatically returned to the algorithm, again via the optimization object. The performance parameters of interest in structural optimization are logically aggregated in a *performance object* (described in Section 4.3.2), and are used

by the algorithm to form an objective function evaluation to determine the next step in the optimization process.

A key reason for using this framework to integrate optimization components is that the structural optimization algorithm and the designer both benefit from a system where design modifications trigger automatic analyses and feedback on structural performance. The designer benefits from the intuition of a real-time simulator, as discussed in chapter 3. The optimization algorithm needs a system which can respond to the changes in design variables that it makes by interpreting the significance of these changes, triggering an analysis of the structure, and returning analysis results and performance evaluations to determine the next step the algorithm should take. The temporal performance of a structural optimization algorithm, in cases where the structural analysis occurs in a separate program, is dependent to an extent on the time it takes to pass data from one program to another. In the integrated framework presented here, this data passing will not rely on file writing to a hard drive, but rather on passing data between objects which are stored in RAM. This is orders of magnitude faster than an approach involving file writing and reading.

4.3.1 Optimization Object

An optimization class is created to serve as an interface between an optimization algorithm and the other classes contained in the implementation of the architecture. The primary task of an optimization object instantiated from this class is to combine all the data and references needed into the optimization problem statement defined by the user, and to specify how changes in the design variables affect the model being optimized.

The optimization object takes a reference to one or more structural models when instantiated. This allows it to access all public attributes of the structural model and of the mathematical models of the structural model. Although the optimization object uses the reference to the models to change design variables, the responses for the objective function typically come from a performance object (section 4.3.2), which contains references to the relevant properties in the models.

4.3.2 Performance Object

In a scenario where multiple models of an artifact exist, each model can estimate the performance of the structure in different ways. This is true of both structural and mathematical models. A given structural model captures a certain amount of information about the structure, and may well ignore features that other models consider. For example, truss models ignore bending effects in members whereas frame models capture them. As another example, a given mathematical model may analyze only the static effects of a structural model, or may also capture the dynamic behavior.

Each model can produce different performance parameters which can be used to evaluate the structure, motivating the management of these parameters in an organized manner so that they can be enumerated and accessed with ease. The performance object retrieves information from the structural models of the artifact, which publish their available performance parameters as a list of publicly accessible properties.

Freedom is given to the user to specify which parameters to include in a particular instance of a performance object, which is useful even without optimization since it allows the user to tailor the environment to prominently display those performance parameters which are of greatest interest. The classification structure below shows a parent performance class which is a generalization of derived specific performance classes.

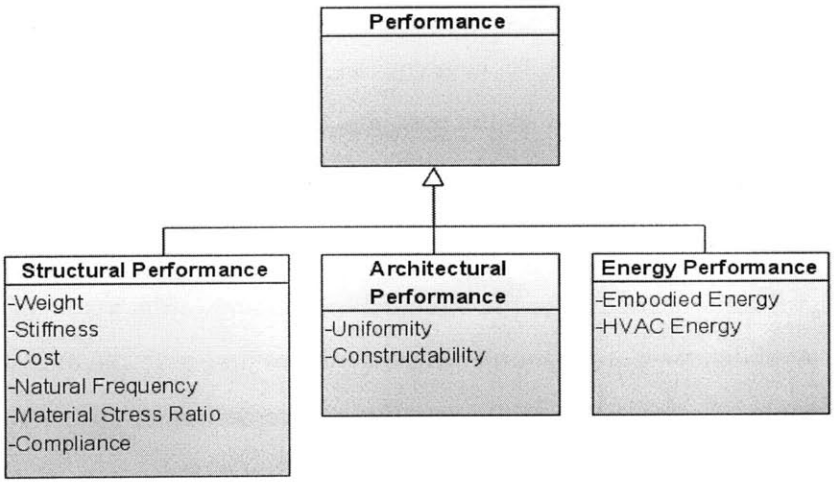


Figure 4.1 OOP classification structure of performance objects

Assuming a model is present in the framework which can evaluate the parameters in the object, the object can be instantiated. The classes shown in Figure 4.1 are examples of the kind of performance parameters that a user may specify. If a user attempts to instantiate a performance object containing a parameter that cannot be calculated, an exception is thrown.

The advantage of the performance object in the context of user-driven design changes is clear. It allows the user to see which parameters are available to him from the connected models, and to specify which of them he wants to observe on the interface. The combined system of performance and optimization objects interacting with the system is shown in the classification diagram of Figure 4.2.

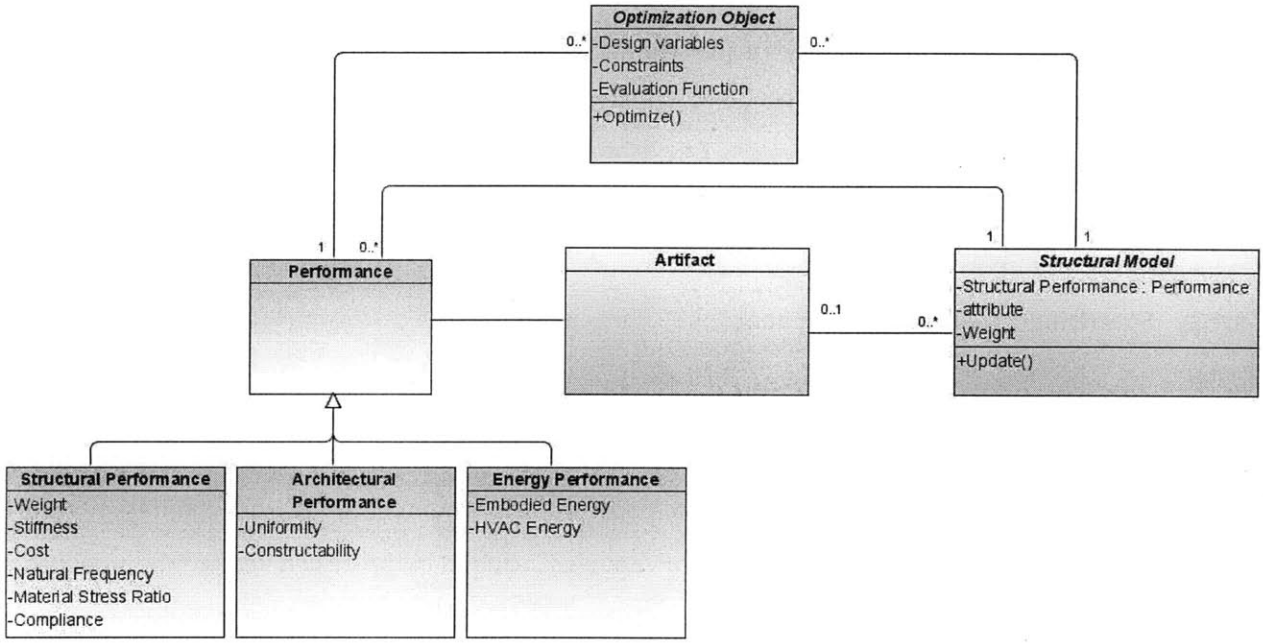


Figure 4.2 Optimization object and performance object in the framework

4.4 Application

The ideas outlined in this chapter are implemented in a program developed using the framework of chapter 3. A major benefit of implementing structural optimization in such a framework is, as discussed previously in this chapter, that new algorithms can easily be added through minor modifications of the optimization object to link it to the new algorithm and configure it appropriately.

The implementation described in this section is relatively simplistic. It is built using Microsoft's C# (Microsoft Corporation, 2010), a fully objected oriented programming language, and takes as its starting point a program containing a single structural model object (a two-dimensional pin-jointed truss model) and a single mathematical model (a small displacement stiffness matrix model). A performance object gathers the cost, compliance, and weight properties of the truss model object as performance evaluation criteria, and a bespoke graphical environment allows for interactive modification of artifact and model objects, rendering, and displaying of all other information required.

The optimization algorithm in this implementation deals only with geometry optimization of existing topologically defined truss models. The algorithm used is contained in the *MinConNLP* class of Visual Numerics' IMSL C# Numerical Library (Visual Numerics, Inc., 2010), and is based on the Sequential Quadratic Programming (SQP) method with an Active Set technique. Further details of the algorithm can be found in Spelluci (1998a; 1998b).

4.4.1 Specifying the Problem Statement

The design vector consists of the nodal coordinates and the cross-sectional areas of the truss members. The user is free to include or exclude any nodal coordinates or member areas from the design vector, and to set upper and lower bounds on all design variables included in the design vector. The screenshot in Figure 4.3 shows a 'Node Properties' window which opens when the user clicks on a node. In this way, the user can set up the optimization problem in the same environment in which modeling and analysis take place. A similar window can be opened to specify which member properties form part of the design vector, and the bounding constraints that should be imposed on them.

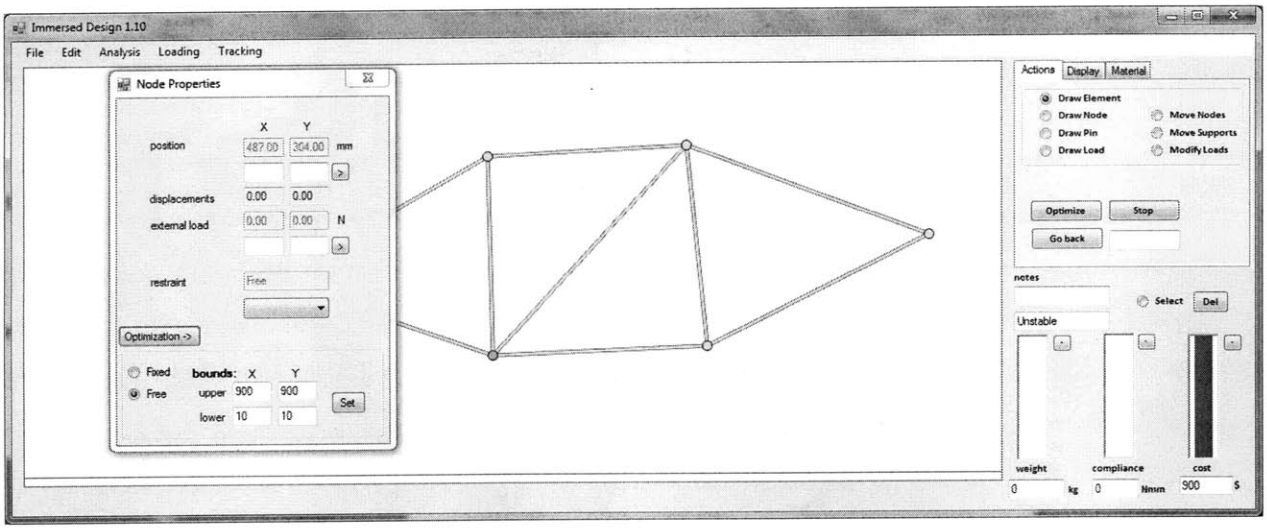


Figure 4.3 Screen shot showing Node Properties window with optimization features

4.4.1.1 Objective and Constraint Functions

The objective and constraint functions are almost always contained in a performance object. For multi objective problems, the weighted sum approach described in 4.2.1 is used. The objective function consisting of k performance parameters $f_i(\mathbf{x})$, for a design vector denoted by **x**, is given by

$$min \sum_{i=1}^{k} w_i f_i(\boldsymbol{x})$$

where w_i is the weighting attached to the i^{th} performance parameter. Note that it is assumed that we seek a minimization of parameters. If this is not the case, the sign of a particular $f_i(\mathbf{x})$ can be changed to convert a minimization problem to a maximization problem. In order to facilitate ease of use, the weights attached to objectives can conveniently be altered using visual controls on the interface, as seen in Figure 4.4.

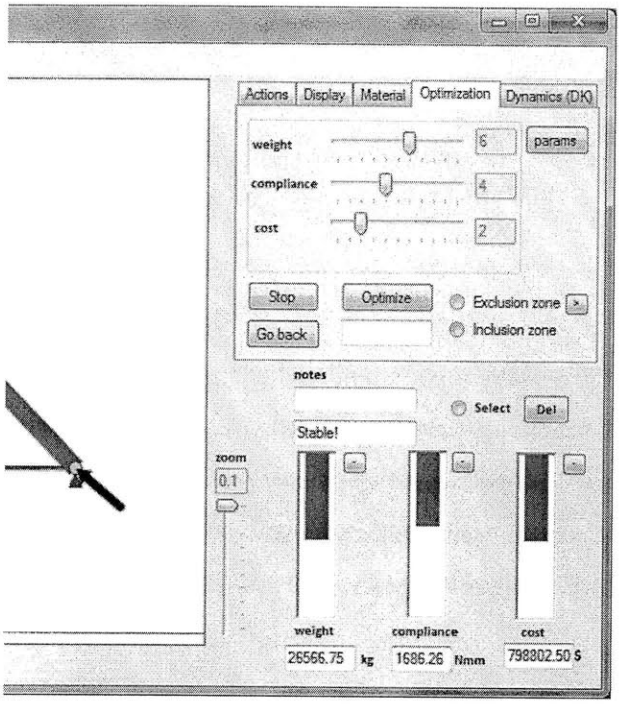


Figure 4.4 Modification of objective function in the GUI

Constraints can be placed on all the design vectors in the form of upper and lower bounds. The nodal coordinates can be further constrained to lie within or outside of certain spaces, as described in 4.4.1.2. Constraints can be placed on performance parameters to prevent them rising above or falling below a certain threshold, as seen in Figure 4.5. An equality constraint is implicitly imposed in the form of equilibrium. If the model is not in stable equilibrium, it returns an exception which is interpreted by the optimization object as a violation of the equality constraint on equilibrium.

4.4.1.2 Exclusion and Inclusion Zones

The user can graphically specify exclusion zones on the interface. These are regions of space in which no node can lie, and are a key feature in allowing the user to control the shapes generated by the optimization algorithm. A similar concept appears in a program based on a shape optimization algorithm, developed by Smith et al. (Smith et al., 2002). The exclusion zones are represented in the problem statement as constraints on the allowable values of a node's coordinates.

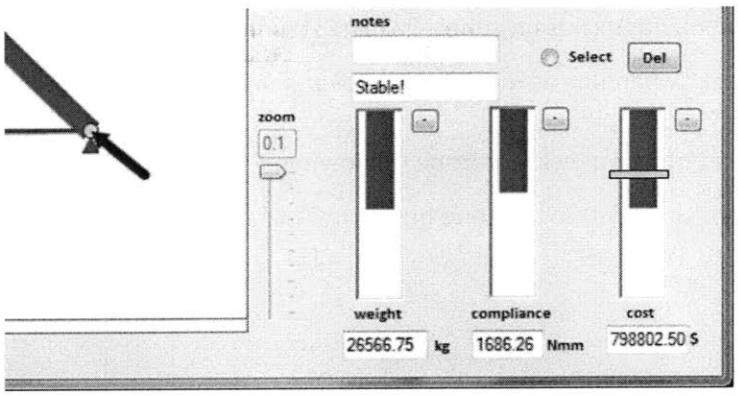


Figure 4.5 Specification of constraints on performance parameters. Yellow line indicates upper bound being set on cost

Each exclusion zone generates two constraints on each of the node's coordinates. In two dimensional space, including an exclusion zone adds four constraints per node to the problem statement. The problem statement can be extended to prevent members connecting nodes from lying within an exclusion zone. The inbuilt shape intersection methods of Microsoft's .NET framework can conveniently check if the line representing a member intersects with the polygon representing the exclusion zone, and thus generate the appropriate constraint evaluations.

Exclusion zones allow the designer to retain some control over the aesthetics of the shape. There are, however, practical as well as aesthetic benefits to retaining control over shape in such a manner. Constraining the structural members to keep certain areas free for non-structural purposes is a commonly encountered requirement in structural design. For example, a bridge must be constrained so that none of its structural members obscure the roadway or, in some cases, a shipping channel under the bridge. The contribution of this approach is to use exclusion zones in a fluid design process rather than in a dedicated structural optimization program. Optimization is not treated as a specialist operation, but rather as a natural part of the design process.

Inclusion zones can also be specified. They are regions of space in which a single node is constrained to lie. At the mathematical level, they are no more than a pair of upper and lower bounds on nodal coordinate design variables. To the designer, they represent a useful way to constrain the shape by imposing spatial boundaries. When an initial design has been arrived at using the environment, a degree of adherence to the shape of this design can be specified with a user with no formal optimization

training. This, combined with exclusion zones, is a key step in making optimization methods more useful to real-world designers by handing as much control over shape as desired back to the designer.

The graphical specification of an inclusion zone is shown in Figure 4.6. The top node of this simple three-bar truss is constrained to lie within the zone during optimization of the geometry of the structure.

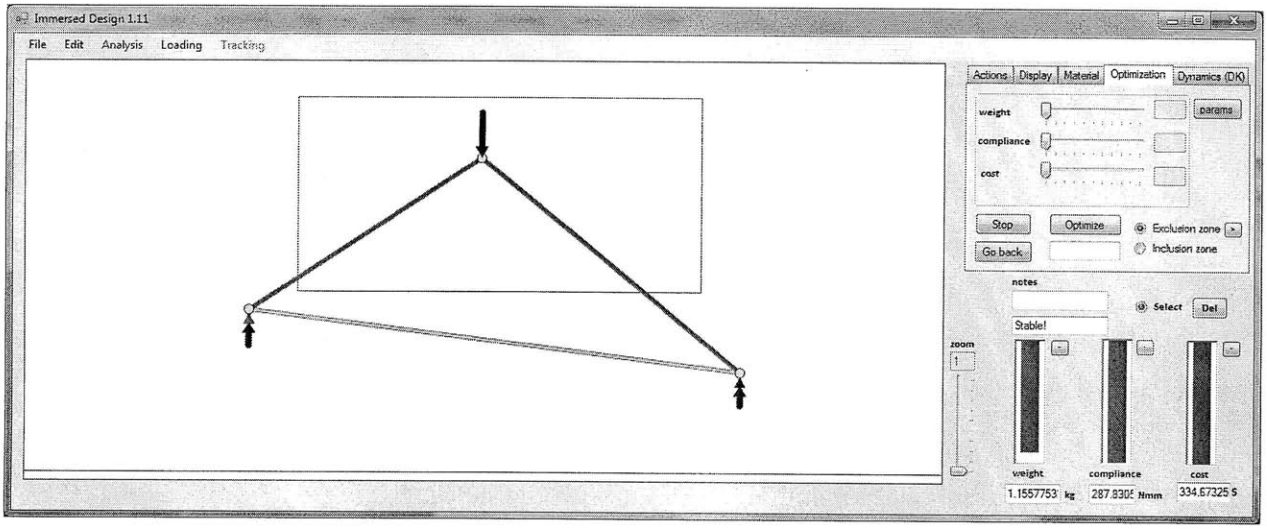


Figure 4.6 Graphical specification of an inclusion zone (represented as red rectangle) in the GUI. Top node is constrained to lie within the zone during optimization

4.4.1.3 Extensibility - Linking New Algorithms to the Software

The interface of the optimization algorithm to the rest of the program resides entirely in the optimization object. Some programming is necessary to add a new optimization algorithm. The advantage to using this framework is that the task of integration is clearly defined for a new user. All variables which could conceivably form part of a design vector are available to the optimization object through its references to the model objects. Information on performance parameters is available through references to a performance object, and structural responses typically appear in the model objects. The developer who wishes to extend the system with a new algorithm must provide the code to assemble the design vector and constraints, and link the algorithm to the optimization object. Everything that is required to do so is conveniently gathered in a single location for the developer.

4.5 Summary

This chapter extended the software framework to allow for the inclusion of optimization algorithms. It proposed solutions to identified limitations in the application of structural optimization to the conceptual design of structures, and described a real implementation of these ideas in a developed piece of software. This chapter showed how optimization should be included in structural design software, and outlined a methodology for researchers wishing to incorporate new or existing algorithms into the framework. It is a step which brings increased relevancy to the field of structural optimization. The entire software framework, including all aspects described in chapters 3 and 4, is presented in Figure 4.7.

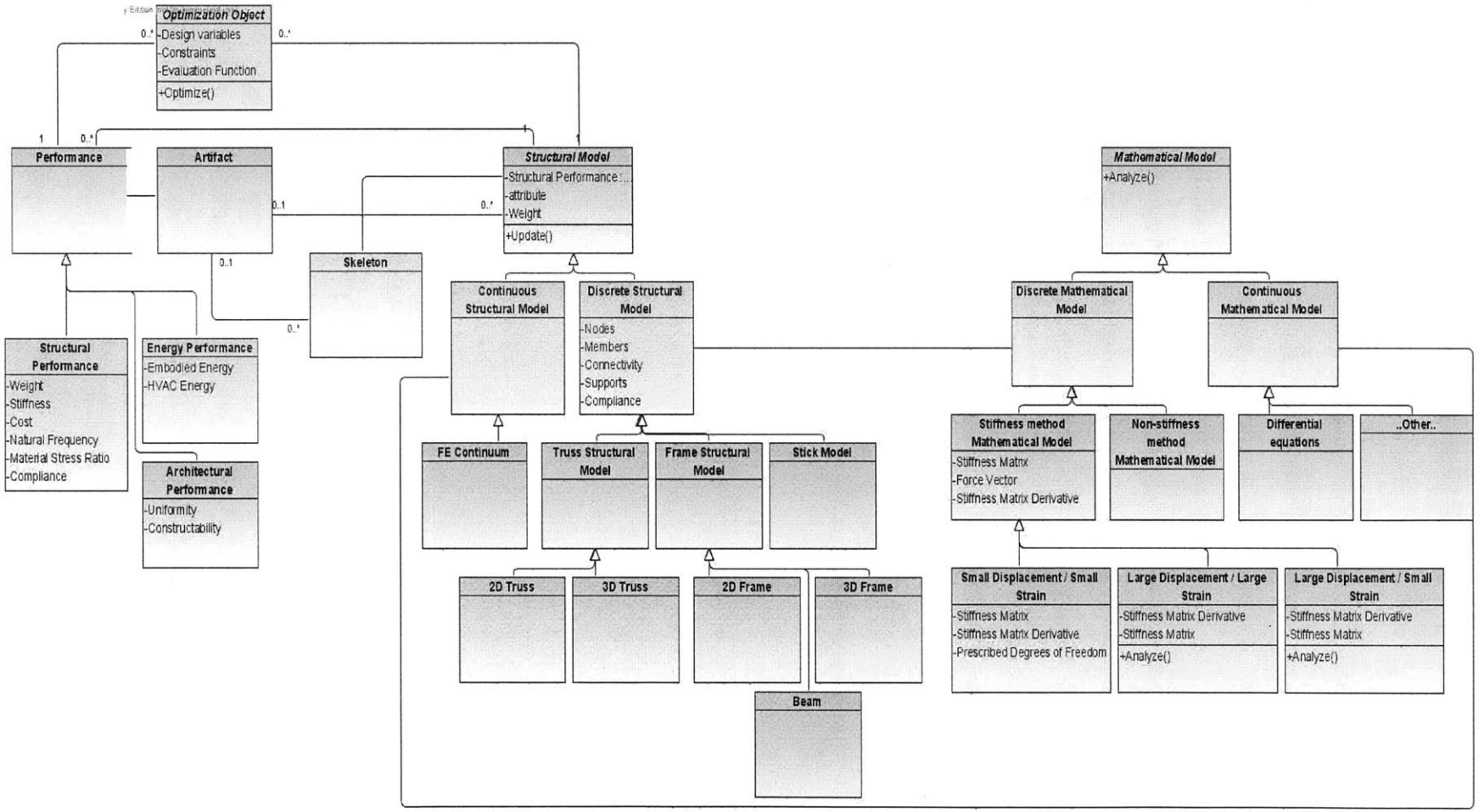


Figure 4.7 Entire software framework, including optimization and performance objects

Chapter 5 Results and Conclusions

5.1 Implementation

To demonstrate the validity of the approach of the thesis to structural design software, a conceptual design program is developed using Microsoft's C# programming language. The program adheres strictly to the framework described in chapters 2 and 3, as can be seen from Figure 5.1, a UML diagram of its classification structure.

A limited range of the possible structural and mathematical models are included. The chosen structural model is a two-dimensional truss, and a small strain stiffness method is included as the mathematical model. A bespoke user interface is created to have full control over the implementation and to allow complete freedom in exploring the possibilities of using this approach to structural software. Event passing and handling in C# is used to trigger analyses and display results in real time. As described in section 4.4, an SQP algorithm is used for shape optimization of trusses. Section 5.2 shows a series of trusses which are designed using the interactive environment of this software.

The software's interface allows users to import images and build structural models on top of them. Engineers could scan in architects' renderings and build structural models appropriately. Students and instructors could use the software to develop a first-order model of the behavior of iconic structures by importing a photograph of them and building a live model in the foreground. While by no means technologically groundbreaking, this is a highly useful and practical feature that is typically not included in structural software. Figure 5.2 shows an imported image of a stone bridge in the software environment. A truss model is used to establish a first-order understanding of the bridge's structural behavior and performance.

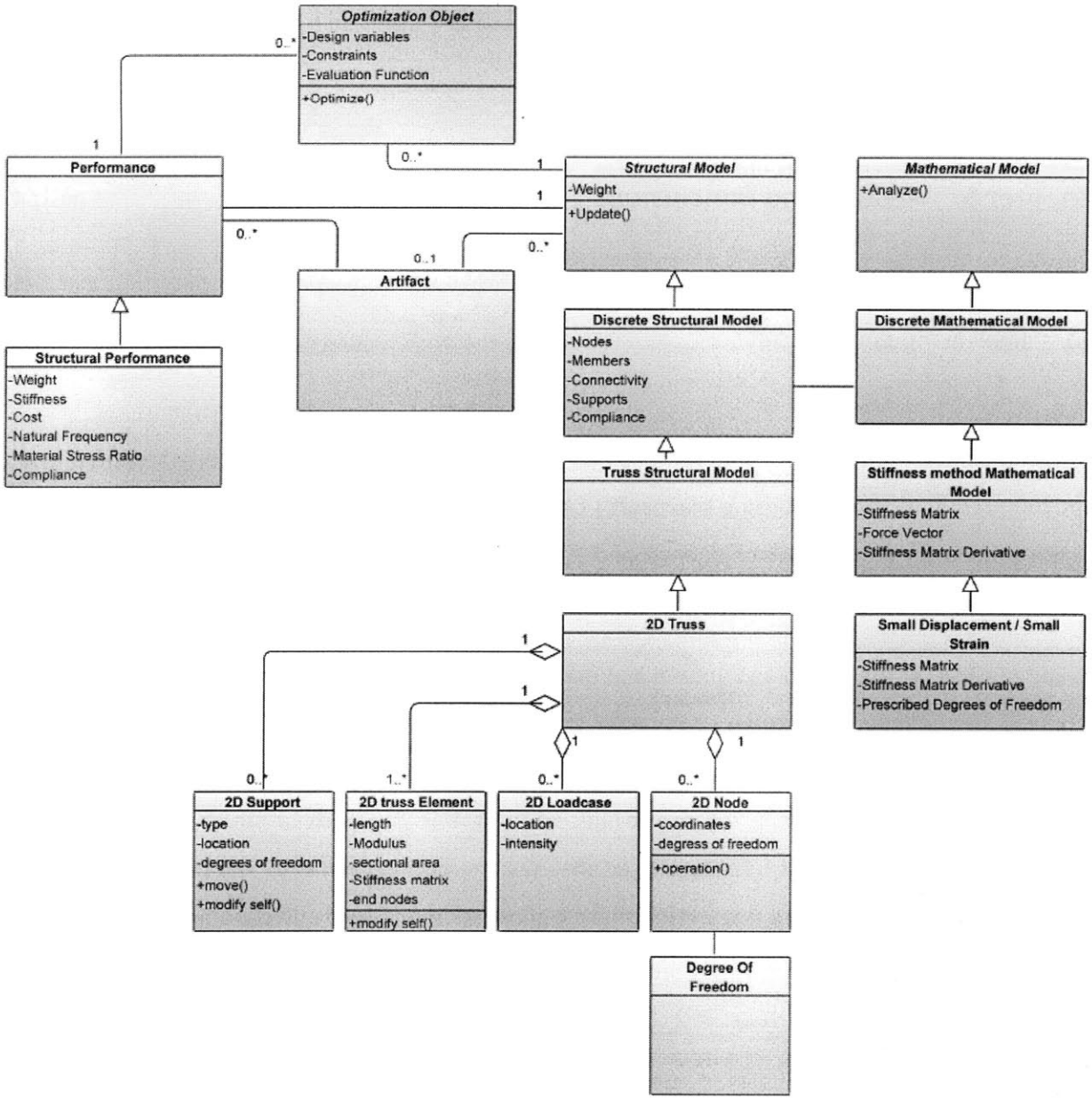


Figure 5.1 UML classification diagram of developed software – an implementation of the software framework of chapters 3 and 4

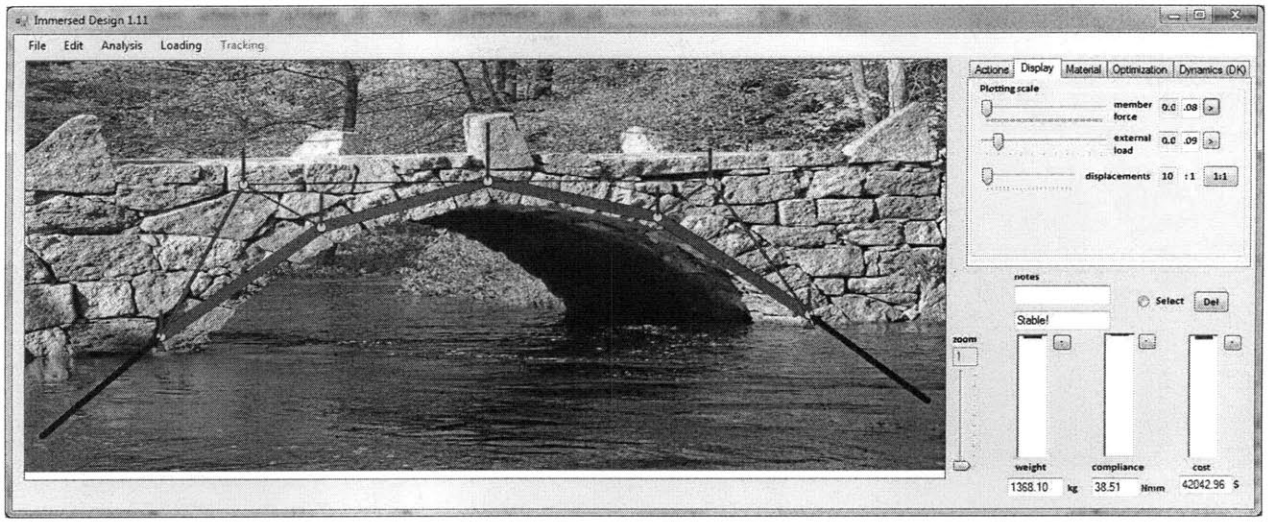


Figure 5.2 Using a truss model for a first-order exploration of the behavior of a stone bridge. All truss member forces are compressive

5.2 Generated Designs

The following series of images shows how a design can be conceptualized and evolved quickly using the software. All analysis results and performance feedback are displayed in real time. Even at this relatively early stage of software development, the simple design exercise below can be completed in well under a minute using *Immersed Design*. There is no need to ask the program to run analyses, as stability checks are constantly running to determine if an analysis is appropriate.

In the following images, the thickness of a member is proportional to the magnitude of force it is carrying. Red members are in compression, while blue members are in tension. Black members, drawn as thin lines, carry no force. Black arrows indicate applied loading, and blue arrows indicate support reactions.

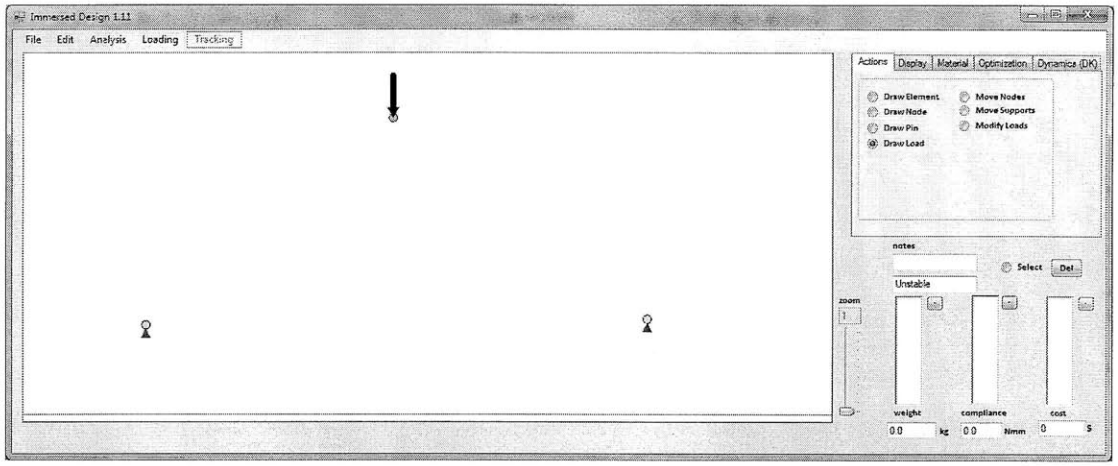


Figure 5.3 Specification of loads to be carried and support conditions – a potential design problem

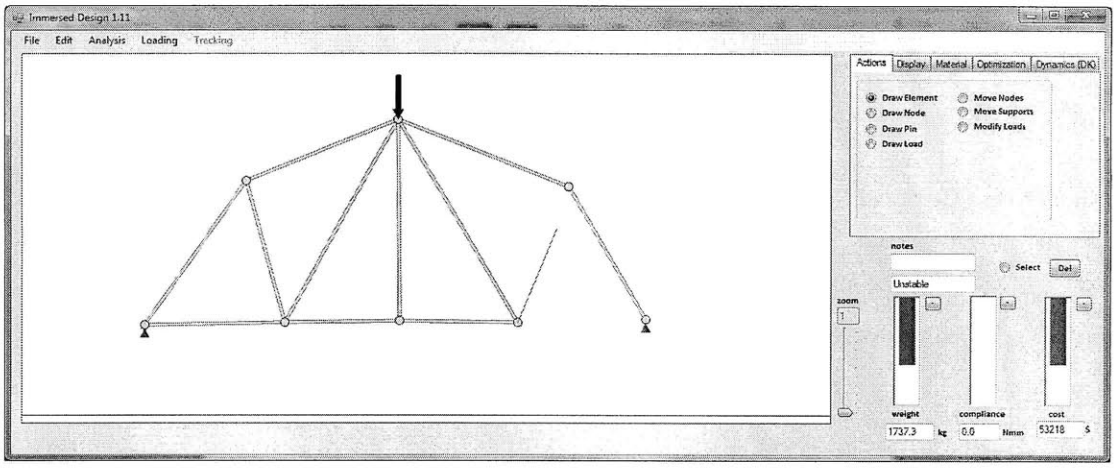


Figure 5.4 An initial outline sketch – stability has not yet been achieved

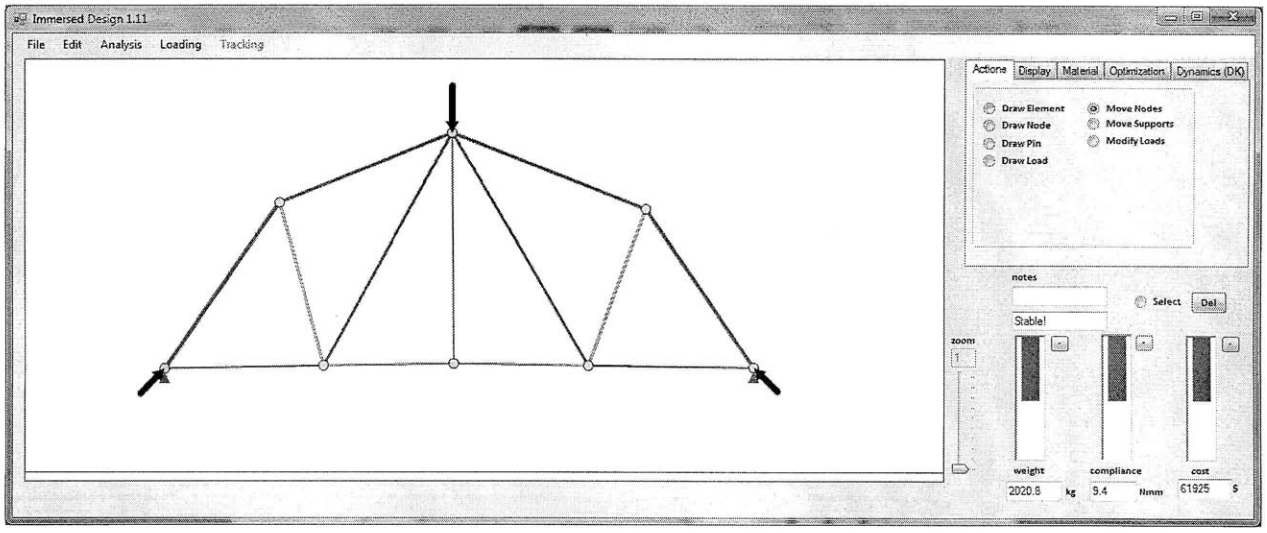


Figure 5.5 Stability condition detected, triggering analysis and feedback

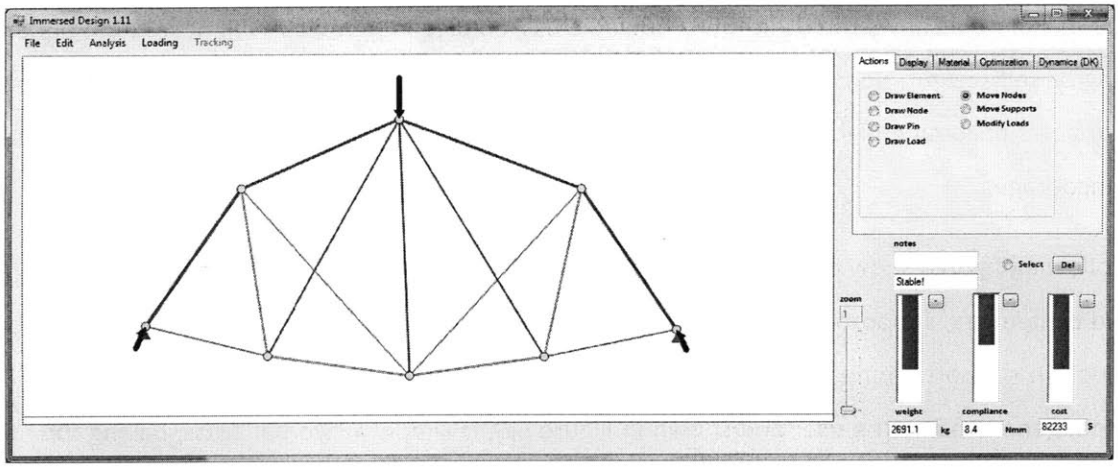


Figure 5.6 Addition of cross-bracing and modification of geometry by the user

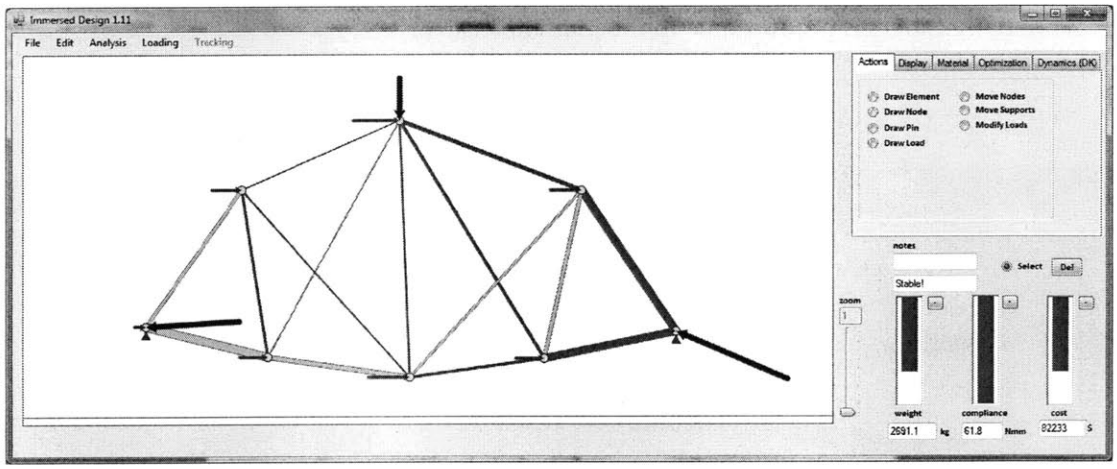


Figure 5.7 Addition of lateral loads to evaluate response to wind loading

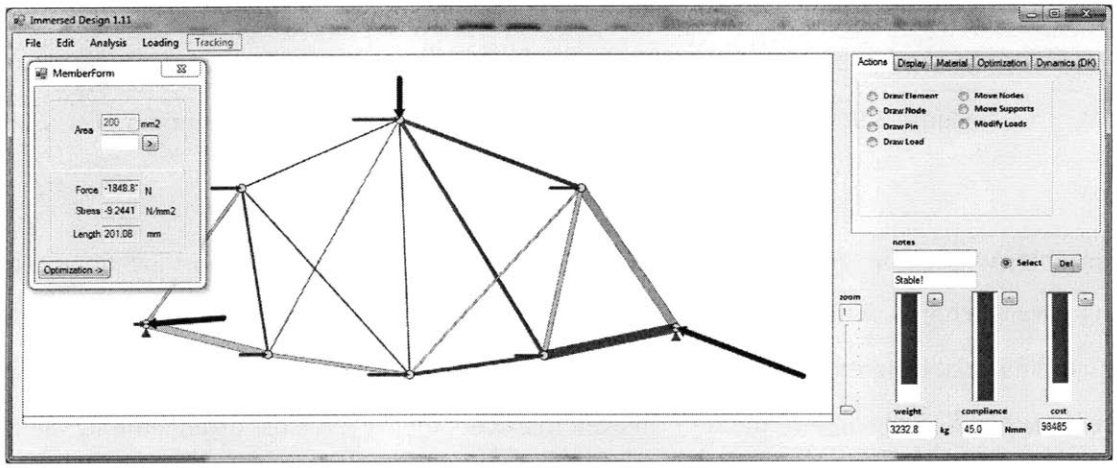


Figure 5.8 Strengthening of members which experience high forces under lateral loading

This brief example shows the user how a bowed truss, an elegant and lightweight solution under gravity loading or self weight, does not perform quite as well under lateral loading, with chord members near the supports becoming highly stressed. This is the kind of discovery that the explorative process allows and encourages.

The following figures show a series of truss designs generated using the software. The trusses are point-loaded cantilevers, supporting a point load that is fixed in magnitude and location. The depth between the two pin supports remains constant, although the depth of the truss itself is free to vary. The first solution, generated by the user and shown in Figure 5.9, is simply a two-bar truss, having the minimum number of members necessary to support the load. The member forces in the bars are impractically large. All examples are statically determinate, although not necessarily so, meaning the required cross-sectional are of members is proportional to the load they carry.

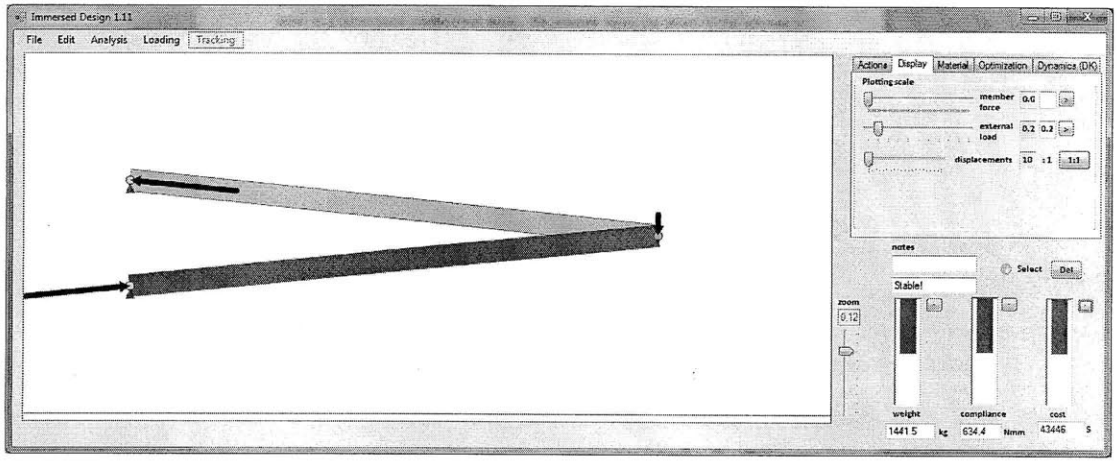


Figure 5.9 Two-bar cantilever truss, with high forces in the members

The design is modified by the user in Figure 5.10. For the same loading and support locations, a strategy of using parallel chords with diagonally-braced, equally-spaced spans is chosen. Compared to the two-bar design, material weight is almost halved, although compliance increases dramatically. In Figure 5.11, the user has modified the arrangement of the web members to give a K-truss arrangement, still with the same equal bay spacing. The fluid nature of the software means that this transformation is easily and quickly achieved, allowing the designer to flexibly explore a wide range of possibilities.

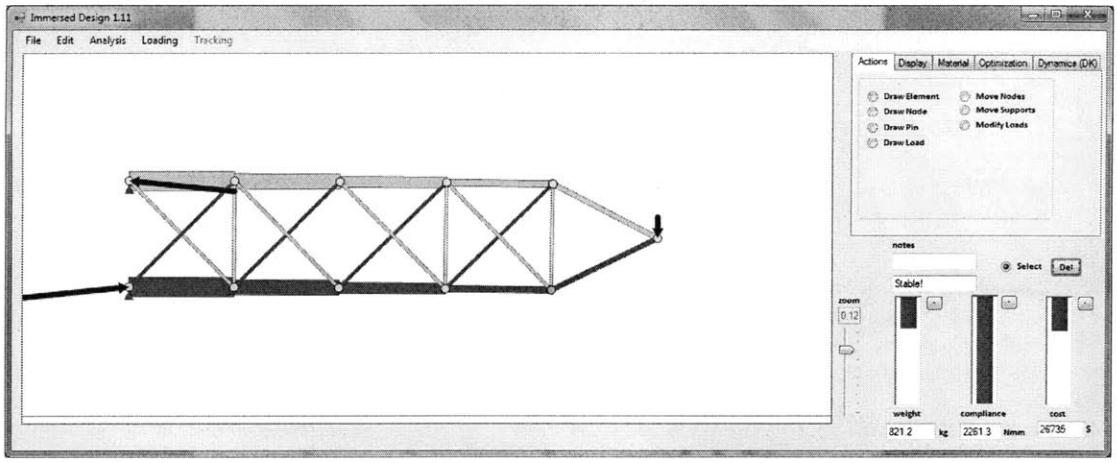


Figure 5.10 Parallel chords with double cross-bracing in each panel

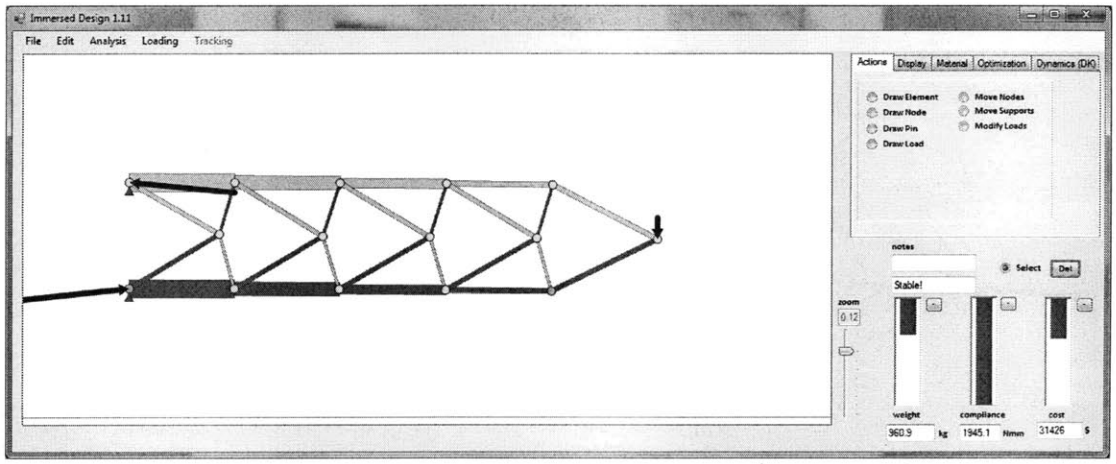


Figure 5.11 User modification of web members to give a K-truss arrangement

Figure 5.12 shows the same solution, having a K-truss arrangement of the web members, but with bowed top and bottom chords. The weight of material required is less than the parallel-chord K-truss, and compliance is reduced substantially. This solution is modified to give the classical Michell truss (Michell, 1904), as shown in Figure 5.13. This is the lightest and stiffest of all solutions found so far. Despite the intricacy of arrangement and multiple connections, which contribute significantly to the estimate of cost, the Michell truss is also cheaper than some previous designs, and not much more expensive than the cheapest.

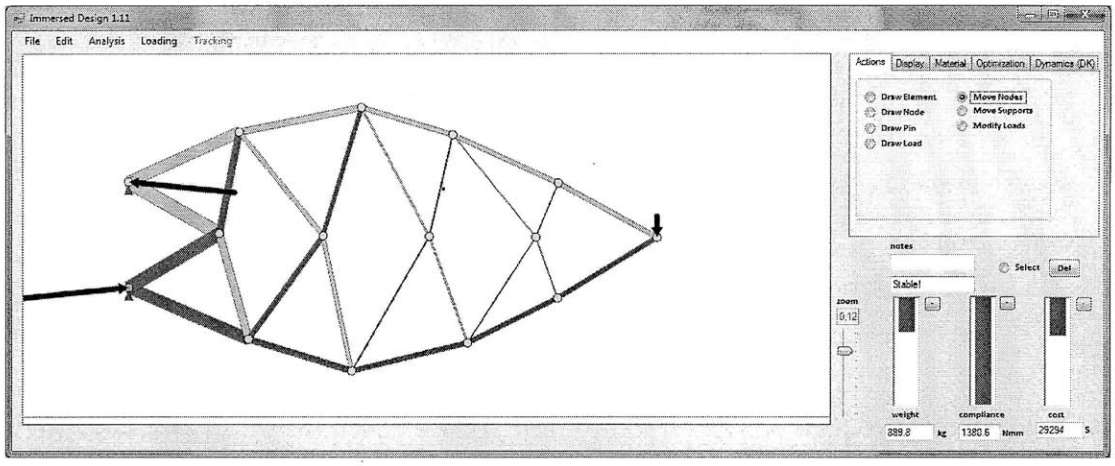


Figure 5.12 User-driven bowing of chords, for a K-truss arrangement of web members

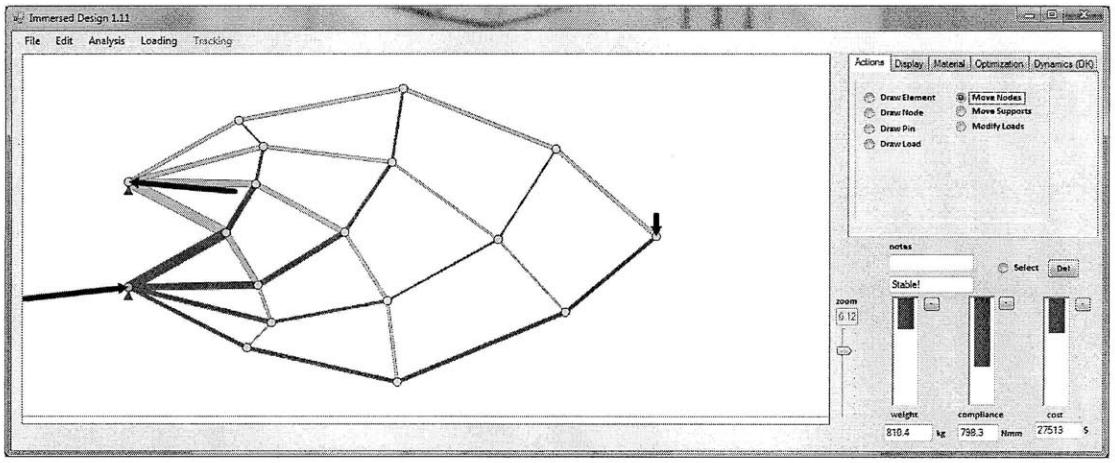


Figure 5.13 Modification of web members to give a classical Michell truss

In order to mitigate the relatively large support reactions, a third pin support is added to the right of the first two. The user could alternatively explore the possibility of increasing the vertical distance between the supports to achieve this reduction. The bowing action of the chords is next reduced to give a shape that is closer to a conventional parallel-chord design. The increase in weight is relatively low, showing that the Michell solution is fairly insensitive to small changes in shape of the outer chord members. This example of an evolved set of designs shows the power of the software in enabling the user to easily and flexibly explore a wide range of design possibilities, while receiving instant feedback on stability, structural responses, and a range of performance metrics.

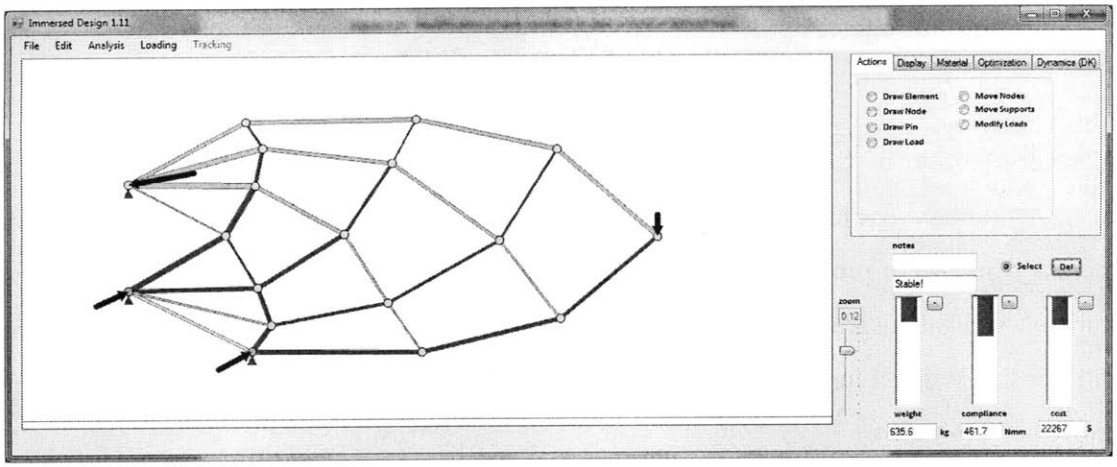


Figure 5.14 Reduction in bowing of chords, and addition of a third pin support to alleviate high concentrated support reactions

5.3 Conclusion

The thesis set out to produce a structural design tool that combines user intuition and experience with mathematical optimization to encourage creative exploration in the seeking of design solutions.

A review of the literature and of existing structural software motivated the development, from scratch, of an object-oriented software framework. This framework makes use of some established principles of good structural mechanics software design, but is developed specifically for structural design software rather than to adapt existing analysis software to a design scenario. This framework enables the customization and extensibility of software developed using it by separating modeling and analysis components at increasing levels of abstraction and by defining interface contracts for any additional components.

A review of the structural optimization literature, with a particular emphasis on application to the conceptual design of structures and how it can be improved, identified areas for improvement. Chapter 4 extended the software framework to incorporate optimization algorithms and to facilitate the inclusion of new algorithms and customization of existing ones. Finally, a real-time structural design tool with optimization functionality was developed using this framework.

5.3.1 Key Contributions

A major contribution of this thesis is the developed interactive immersed design environment for structures. Although basic in comparison to how this software will look when developed to a commercial standard, it provides all the functionality described to the design of basic truss and frame structures. It is highly useful for developing structural intuition and bringing physical considerations to the initial design stage. It further encourages an exploration of the design space through its optimization elements.

It is a fully functional teaching tool for structural design, useful for practicing designers to explore how simple structures perform, and a powerful demonstration of the merits of this new approach to structural design.

The second contribution is the framework for structural design software developed in chapter 3. This is a unique and original approach to the problem, representing a major change in the way structural design software is built. If widely adopted, the approach could significantly change the way design is performed in practice.

The third contribution is the relevancy and rejuvenation that the work brings to the field of structural optimization which, despite its many strengths, is not significantly applicable at present to the building industry and the world of structural design. The work rethinks the role that optimization plays in the conceptual design phase, and specifies exactly how algorithms should integrate with design software in the future to best fulfill this role.

5.3.2 Future Work

There are a number of required extensions to the software framework. Links should be developed between the multiple structural models of the artifact so that changes to any of the models, or to the artifact, can automatically update all the other models. This is a challenging field of research which would draw on the area of model semantics. The interfaces of the framework described in section 2, which allow for extensibility by other developers, need to be further developed. At present they cover only a small fraction of all classes that a developer may wish to add.

The optimization components should be extended to incorporate elements of topology optimization so that the software can be used to generate candidate designs from scratch, rather than only to modify existing ones. An issue of key importance in accommodating designers with little formal knowledge of optimization is the definition of a strategy to control scaling, tolerances, and other factors to which optimization problems can be highly sensitive.

The design tool itself needs more development to make it applicable to real-world design problems. Having carried out the identified work on the framework, the environment should be extended to include more modeling methods and optimization algorithms. Building codes and material databases could be incorporated. A strategy to work with many designs simultaneously, using fuzzy logic to combine and vary them, is envisaged as a credible alternative to deterministic optimization at the very early stages of conceptual design. The software should thus be extended to enable the designer to conveniently store multiple designs in a small patch on the interface. These could then be picked up and modified (either by the user or by an optimization algorithm) in real time.

References

Abdalla, R.S. and John Yoon, C. (1992) 'Object oriented finite elements and graphics data translation facility', *Journal of Computing in Civil Engineering*, vol. 6, pp. 302-322.

Afonso, S.M.B., Lyra, P.R.M., Albuquerque, T.M.M. and Motta, R.S. (2010) 'Structural analysis and optimization in the framework of reduced-basis method', *Structural and Multidisciplinary Optimization*, vol. 40, no. 1-6, pp. 177-199.

Akira, W. (1999) 'Considering structural design and analysis in computer age', *Kenchiku Gijutsu*, no. 593, pp. 115-117.

Altair Engineering (2010) *Altair Optistruct*, [Online], Available: http://www.altairhyperworks.com/(S(4tynfy553r21ikbhqr4u3w55))/Product,19,OptiStruct.aspx [11 February 2010].

Arcade (2002), [Online], Available: http://www.arch.virginia.edu/arcade/ [9 February 2009].

Bendsoe, M.P. (1995) *Optimisation of Structural Topology, Shape and Material*, Berlin : Springer-Verlag.

Bolton, A. (2005) 'Differential Thermal Expansion Blamed for Paris Terminal Collapse', *New Civil Engineer*, March, [Online], Available: http://www.nce.co.uk/differential-thermal-expansion-blamed-for-paris-terminal-collapse/534949.article [30 Mar 2010].

Booch, G. (2007) *Object-Oriented Analysis and Design with Applications*, Upper Saddle River, NJ: Addison-Wesley.

Chilton, J. (2000) *Heinz Isler: The Engineer's Contribution to Contemporary Architecture* , London: Thomas Telford Publishing.

Clune, R., Connor, J. and Ochsendorf, J. (2009) *'An Interactive Optimization Tool for Structural Design', 10th US National Conference on Computational Mechanics, Columbus, OH*, [Online], Available: http://web.mit.edu/clune/www/USNCCM10/ [12 May 2010].

Coello Coello, C.A. (1999) 'A comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques', *Knowledge and Information Systems*.

Cohn, M.Z. (1994) 'Theory and practice of structural optimization', *Structural Optimization*, vol. 7, pp. 20-31.

Computers & Structures Inc. (2010) *CSI: Computers & Structures Inc.*, [Online], Available: http://www.csiberkeley.com/products_SAP.html [10 February 2010].

Connor, J.J. (2003) *Introduction to Structural Motion Control*, Upper Saddle River, NJ: Prentice Hall.

Cross, J.T., Masters, I. and Lewis, R.W. (1999) 'Why you should consider using object-oriented programming techniques for finite element methods', *International Journal of Numerical Methods for Heat & Fluid Flow*, vol. 9, no. 3, pp. 333-347.

Dassault Systemes (2002) *CATIA - Virtual Design for Product Excellence*, [Online], Available: http://www.3ds.com/products/catia/welcome/ [10 February 2010].

Dassault Systemes (2004) *Abaqus FEA*, [Online], Available: http://www.simulia.com/products/abaqus_fea.html [10 February 2010].

Deb, K. and Gulati, S. (2001) 'Design of truss structures for minimum weight using genetic algorithms', *Finite Elements in Analysis and Design*, vol. 37, pp. 447-465.

Delalondre, F., Smith, C. and Shephard, M.S. (2010) 'Collaborative software infrastructure for adaptive multiple model simulation', *Computer Methods in Applied Mechanics and Engineering*, vol. 199, no. 21-22, pp. 1352-1370.

Dr. Software (1998), [Online], Available: http://www.drsoftware-home.com/ [9 February 2009].

Dubois-Pelerin, Y. and Regon, P. (1998) 'Object Oriented Programming in Nonlinear Finite Element Analysis', *Computers and Structures*, vol. 67, pp. 225-241.

Floudas, C.A. and Pardalos, P.M. (ed.) (2003) *Frontiers in Global Optimization*, Dordrecht, The Netherlands: Kluwer Academic Publishers.

Ford, A. (2009) *Modeling the Environment*, Washington D.C.: Island Press.

Forde, B.W.R., Foschi, R.0. and Stiemer, S.F. (1990) 'Object oriented finite element analysis', *Computers and Structures*, vol. 34, pp. 355-374.

Fox, R.L. (1971) *Optimization methods for engineering design*, Reading, MA: Addison-Wesley.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995) *Design Patterns - Elements of Reusable Object-Oriented Software*, Reading, MA: Addison-Wesley.

Gero, J.S. (1994) 'Towards a model of exploration in computer-aided design', in Gero, J.S. and Tyugu, E. *Formal design methods for CAD*, Amsterdam: North Holland Publishing Co.

Greenwold, S. (2003) *Active Statics*, [Online], Available: http://acg.media.mit.edu/people/simong/statics/ [10 February 2009].

Guest, J. and Igusa, T. (2008) 'Structural optimization under uncertain loads and nodal locations', *Computer Methods in Applied Mechanics and Engineering*, vol. 198, pp. 116-124.

Haftka, R. and Gurdal, Z. (1992) *Elements of Structural Optimization*, Dordrecht, The Netherlands: Kluwer Academic Publishers.

Hemp, W.S. (1973) *Optimum Structures*, Oxford: Clarendon.

Heng, B.C.P. and Mackie, R. (2009) 'Using design patterns in object-oriented finite element programming', *Computers and Structures*, vol. 87, pp. 952-961.

Heyman, J. (1959) 'On the absolute minimum weight design of framed structures', *The Quarterly Journal of Mechanics and Applied Mathematics*, vol. 12, no. 3, pp. 314-324.

Imai, K. and Schmit, L.A. (1981) 'Configuration optimization of Trusses', *ASCE, Journal of the Structural Division*, vol. 107, no. 5, pp. 745-756.

Kemp and Ueki-Polet (2009) *Less and More: The Design Ethos of Dieter Rams*, Osaka: Suntory Museum.

Kirsch, U. (1989) 'Optimal topologies of truss structures', *Computer Methods in Applied Mechanics and Engineering*, vol. 72, no. 1, pp. 15-28.

Larman, C. (2004) *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, Upper Saddle River, NJ: Prentice Hall.

Lawson, B. (1980) *How Designers Think*, Westfield, NJ: Eastview Editions.

Lindblad, A. and Miller, G.R. (2004) 'Generation, analysis and comparison of multiple structural design alternatives in an interactive real-time environment', *Engineering with Computers*, vol. 19, p. 255–263.

Mackie, R. (1992) 'Object oriented programming of the finite element method', *International Journal of Numerical Methods in Engineering*, vol. 35, pp. 425-436.

Mackie, R. (1998) 'An object-oriented approach to fully interactive finite element software', *Advances in Engineering Software*, vol. 29 , no. 2, pp. 139-149.

Mackie, R. (2001a) 'Object Oriented Programming for Structural Mechanics: A Review', in Topping, B.H.V. (ed.) *Civil and structural engineering computing*, Stirling, UK: Saxe-Coburg Publications.

Mackie, R. (2001b) *Object-oriented methods and finite element analysis*, Stirling, UK: Saxe-Coburg Publications.

Martini, K. (2001) 'Non-linear Structural Analysis as Real-Time Animation: Borrowing from the Arcade,', Proceedings of the Computer-Aided Architectural Design Futures 2001 Conference, Dordrecht, The Netherlands, 643-656.

McNeel (2007) *Rhinoceros - NURBS Modeling for Windows*, [Online], Available: http://www.rhino3d.com/ [24 March 2010].

Menterey, P. and Zimmerman, T. (1993) 'Object Oriented Non-Linear Finite Element Analysis: Application to J2 Plasticity', *Computers and Structures*, vol. 5, pp. 767-777.

Michell, A.G.M. (1904) 'The limits of economy of material in framed structures', *Philosophical Magazine*, vol. 8, no. 47, pp. 589-597.

Microsoft Corporation (2010) *The C# Language*, [Online], Available: http://msdn.microsoft.com/en-us/vcsharp/aa336809.aspx [5 Jun 2010].